PAOLO CAMAGNI RICCARDO NIKOLASSY
EMANUELE FALZONE

SVILUPPARE APP PER ANDROID

Quaderni di tecnologie

EDITORE ULRICO HOEPLI MILANO

Quaderni di tecnologie

La collana Quaderni di tecnologie è costituita da volumi monografici che trattano singolarmente argomenti specifici inerenti a tecnologie di ambito informatico, elettrico, elettronico, meccanico e meccatronico.

Lo scopo della collana è fornire al docente strumenti didattici specifici per singoli argomenti al fine di realizzare o integrare un percorso formativo adatto al proprio piano di lavoro.

L'approccio didattico vuole essere estremamente "semplificato" senza essere banale, mantenendo rigorosità nella terminologia ma essenzialità negli aspetti teorici privilegiando l'attività laboratoriale.

I Quaderni sono quindi uno strumento didattico realmente duttile, che consente al docente di costruirsi percorsi di insegnamento su misura, combinando argomenti/temi in funzione delle proprie specifiche esigenze.

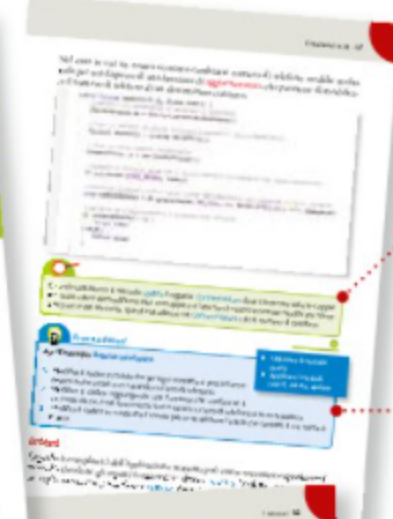
L'elenco completo dei titoli disponibili è riportato all'indirizzo web www.hoepliscuola.it.

Presentazione del volume

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. basato sul kernel Linux ma dove la quasi totalità delle utilities sono sostituite da software scritto in linguaggio Java. È un sistema operativo progettato principalmente per dispositivi mobili quali smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass), che nel 2015 si attestava ad una quota di mercato mobile di circa l'82%.

Android Studio è un ambiente integrato (IDE) per lo sviluppo di applicazioni per la piattaforma Android che si sta diffondendo ampiamente nel mondo dei programmatori per dispositivi mobili. Android Studio consente allo sviluppatore di programmare in maniera più agevole rispetto ai predecessori (Eclipse e Netbeans), oltre ad essere molto stabile. Il volume, attraverso esempi pratici coinvolgenti e assai semplici da comprendere, tratta i tre aspetti essenziali dello sviluppo di app per Android:

- gli elementi grafici di programmazione (widget);
- i sensori;
- i database.

Struttura dell'opera

L'eBook+ che completa il volume presenta l'intero testo in versione digitale, utilizzabile su tablet, LIM e computer, e offre alcuni contenuti aggiuntivi.

Indice

L'OFFERTA DIDATTICA **HOEPLI**

L'edizione **Openschool** Hoepli offre a docenti e studenti tutte le potenzialità di Openschool Network (ON), il nuovo sistema integrato di contenuti e servizi per l'apprendimento.

Edizione **OPENSCHOOL**

LIBRO DI TESTO

Il libro di testo è l'**elemento cardine** dell'offerta formativa, uno strumento didattico **agile** e **completo**, utilizzabile **autonomamente** o in combinazione con il ricco **corredo digitale** offline e online. Secondo le più recenti indicazioni ministeriali, volume cartaceo e apparati digitali **sono integrati in un unico percorso didattico**. Le espansioni accessibili attraverso l'eBook+ e i materiali integrativi disponibili nel sito dell'editore sono puntualmente richiamati nel testo tramite apposite icone.

eBOOK+

L'**eBook+** è la versione digitale e interattiva del libro di testo, utilizzabile su **tablet**, **LIM** e **computer**. Aiuta a comprendere e ad approfondire i contenuti, rendendo l'apprendimento più attivo e coinvolgente. Consente di leggere, annotare, sottolineare, effettuare ricerche e accedere direttamente alle numerose **risorse digitali integrative**.

➔ Scaricare l'eBook+ è molto **semplice**. È sufficiente seguire le istruzioni riportate nell'ultima pagina di questo volume.

RISORSE ONLINE

Il sito della casa editrice offre una ricca dotazione di **risorse digitali** per l'approfondimento e l'aggiornamento. Nella pagina web dedicata al testo è disponibile **MyBookBox**, il contenitore virtuale che raccoglie i materiali integrativi che accompagnano l'opera.

➔ Per accedere ai materiali è sufficiente registrarsi al sito **www.hoepliscuola.it** e inserire il codice coupon che si trova nella terza pagina di copertina. **Per il docente** nel sito sono previste ulteriori risorse didattiche dedicate.

PIATTAFORMA DIDATTICA

La **piattaforma didattica** è un ambiente digitale che può essere utilizzato in modo duttile, a misura delle esigenze della classe e degli studenti. Permette in particolare di **condividere contenuti** ed **esercizi** e di partecipare a **classi virtuali**. Ogni attività svolta viene salvata sul **cloud** e rimane sempre disponibile e aggiornata. La piattaforma consente inoltre di consultare la versione online degli eBook+ presenti nella propria libreria.

➔ È possibile accedere alla piattaforma attraverso il sito **www.hoepliscuola.it**.

LEZIONE 1

IL SISTEMA OPERATIVO ANDROID

In questa lezione impareremo

- il concetto di Applicazione
- ad analizzare i vari componenti
- il ciclo di vita di una activity

Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. e basato su kernel Linux.

DEFINIZIONE

Il kernel costituisce il nucleo di un sistema operativo e possiede il compito di fornire, ai processi in esecuzione, l'accesso all'harware.

È stato progettato principalmente per smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass) e altri.

Android TV

Android Auto

Android Wear

Google Glass

Esistono numerose applicazioni, chiamate apps, sviluppate per questa piattaforma e, probabilmente, molte ancora compariranno in tempi futuri.

DEFINIZIONE

Il termine app è una abbreviazione della parola app(lication). Si tratta di programmi eseguiti dall'utente tramite l'interfaccia grafica del terminale Android. Le app sono di tipo Event Driven, cioè guidate dagli eventi gestiti all'interno del dispositivo mobile, come ad esempio il touch dello schermo, le azioni dei sensori ecc. Possono essere di due tipi: Applicazioni vere e proprie, che occupano tutto lo schermo principale come per esempio il browser web standard di Android, oppure Widget che occupano una piccola e fissata porzione dello schermo principale come per esempio l'orologio standard di Android.

La struttura di un'applicazione Android

Le applicazioni Android sono composte da quattro componenti fondamentali:

- Activity
- Service
- Broadcast receiver
- Content provider

Le applicazioni sono formate da uno o più di questi elementi, tuttavia ciascuna app contiene almeno una activity.

Activity

Le activity sono l'elemento fondamentale delle applicazioni, rappresentano il blocco di codice che interagisce con l'utente utilizzando lo schermo e i dispositivi di input messi a disposizione dal dispositivo. Utilizzano componenti GUI, come ad esempio pulsanti, caselle di testo, pulsanti radio ecc., presenti nel package android.widget. Le activity sono probabilmente il modello più diffuso in Android, e vengono create ereditando la classe android.app.Activity.

Service

I service sono programmi che vengono eseguiti in background e non interagiscono direttamente con l'utente. Un servizio può ad esempio riprodurre un brano Mp3,

oppure leggere segnali dai sensori GPS, mentre l'utente utilizza delle activity per fare altre operazioni. Un servizio si realizza estendendo la classe android.app.Service.

Broadcast Receiver

Un Broadcast Receiver viene utilizzato quando si deve intercettare un particolare evento di sistema, come ad esempio quando si scatta una foto o quando parte la segnalazione di batteria scarica. La classe da estendere è android.content.BroadcastReceiver.

Content Provider

I Content Provider sono utilizzati per esporre dati e informazioni. Costituiscono un canale di comunicazione tra le differenti applicazioni installate nel sistema. Si può creare un Content Provider estendendo la classe astratta android.content.Content Provider.

ZOOM

Activity e Applicazioni • Spesso questi due concetti vengono confusi. In generale:
- Activity: sono associate a una singola e ben precisa attività che l'utente può svolgere, come ad esempio selezionare una data da un Time Picker, oppure inserire un nome in una casella di testo.
- Applicazione: contengono delle activity, oltre ad altri elementi, come ad esempio l'applicazione Appunti possiede una specifica activity per modificare una nota oppure una activity per gestire la data di un appuntamento.

Il ciclo di vita di una activity

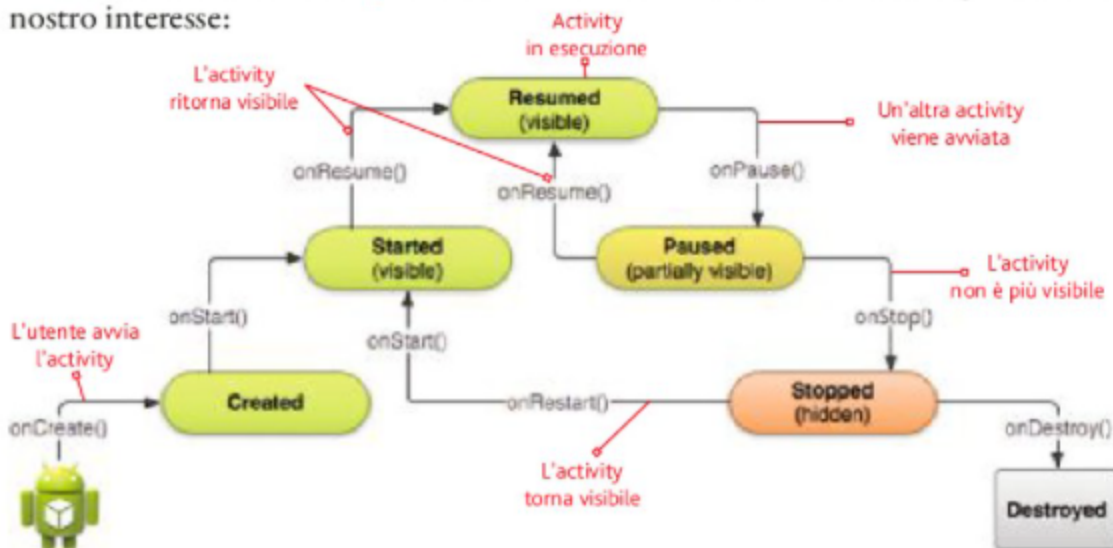
I dispositivi mobili per i quali è stato creato il sistema operativo Android non possiedono schermi capienti come quelli dei PC, per cui le finestre di esecuzione dei programmi non possono essere affiancate o sovrapposte. Le activity Android sono una sorta di programma in esecuzione con delle caratteristiche peculiari, tra cui quella di essere esclusivamente proprietarie del display. In generale una activity passa attraverso i seguenti stati:
- ACTIVE o RUNNING: è visibile ed è in grado di ricevere dati in input;
- PAUSED: è parzialmente visibile, non riceve nessun input;
- STOPPED: non è visibile, tuttavia ancora in esecuzione;
- DESTROYED: è rimossa dalla memoria del dispositivo mobile.

Possiamo eseguire più attività simultaneamente, ma solo una potrà prendere il controllo del display che, come abbiamo detto prima è una risorsa esclusiva. L'attività

che occupa il display è in esecuzione e interagisce direttamente con l'utente, altre attività ancora in esecuzione vengono, per così dire, ibernate in background, per ridurre al minimo il consumo delle risorse del sistema, in questo caso della risorsa elaborazione da parte del microprocessore. L'utente può ripristinare un'attività ibernata e riprenderla da dove l'aveva interrotta, riportandola in primo piano. L'attività dalla quale si sta allontanando verrà ibernata e mandata in sottofondo al posto di quella ripristinata.
Il cambio di attività può anche avvenire a causa di un evento esterno, come ad esempio nel caso di una telefonata in arrivo: se il telefono squilla mentre si sta usando la calcolatrice, quest'ultima sarà automaticamente ibernata e posta in background, l'utente, conclusa la chiamata, potrà richiamare l'attività interrotta e riportarla in vita, riprendendo i calcoli esattamente da dove li aveva interrotti (resumed).

Le attività ibernate non appesantiscono il microprocessore, in quanto non richiedono elaborazione. Pertanto il sistema operativo non mostra all'utente le attività ibernate. Le attività di Android infatti non mettono a disposizione un pulsante di chiusura con cui terminarne l'esecuzione. L'utente può pertanto solo mandare un'attività in ibernazione.

La rimozione delle attività può avvenire spontaneamente, perché hanno terminato i loro compiti, oppure da parte del sistema operativo che decide che non è più utile oppure non ha più memoria a disposizione.

Vediamo quali sono i principali passaggi di stato di un'attività analizzando alcuni metodi della classe Activity che possiamo ridefinire per intercettare gli eventi di nostro interesse:

Analizziamo i principali metodi della classe Activity in modo dettagliato:

protected void onCreate(android.os.Bundle savedInstanceState)
Viene richiamato alla creazione dell'attività, l'argomento savedInstanceState restituisce al metodo un eventuale stato dell'attività passato da un'altra istanza che è stata terminata, l'argomento vale null qualora non vi sia alcuno stato precedentemente salvato.

protected void onRestart()
Viene richiamato per segnalare che l'attività è stata riavviata dopo essere stata precedentemente arrestata.

protected void onStart()
Viene richiamato per segnalare che l'attività viene resa visibile sullo schermo.

protected void onResume()
Viene richiamato per segnalare che l'attività inizia l'interazione con l'utente.

protected void onPause()
Viene richiamato per segnalare che l'attività termina l'interazione con l'utente.

protected void onStop()
Viene richiamato per segnalare che l'attività non è più visibile sullo schermo.

protected void onDestroy()
Viene richiamato per segnalare che l'applicazione è stata terminata.

Per poter modificare il codice di questi metodi dobbiamo eseguire un override del metodo della classe madre, prestando attenzione a inserire nella prima riga di codice di ciascuno di questi metodi il costruttore della classe base che stiamo ridefinendo attraverso l'operatore super.

MainActivity.java

```
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Comprendere il concetto di applicazione (Apps)	○	○	○
Conoscere il ciclo di vita di una activity	○	○	○
Comprendere il ruolo del Sistema Operativo Android	○	○	○
Distinguere gli elementi che rappresentano una activity	○	○	○
Riconoscere gli elementi di una applicazione Android	○	○	○

Competenze	Scarso	Medio	Ottimo
Riconoscere il campo di applicazione di una activity	○	○	○
Riconoscere il campo di applicazione di una Service	○	○	○
Riconoscere il ruolo di una Broadcast Receiver	○	○	○
Riconoscere il campo di applicazione di una Content Provider	○	○	○
Comprendere la fase di ACTIVE e RUNNING di una activity	○	○	○
Comprendere la fase di PAUSED di una activity	○	○	○
Comprendere la fase di STOPPED di una activity	○	○	○
Comprendere la fase di DESTROYED di una activity	○	○	○

2

LEZIONE

ANDROID STUDIO

In questa lezione impareremo

- a conoscere il significato di IDE
- a scaricare e installare Android Studio

Android Studio

Android Studio è un ambiente di sviluppo (IDE) progettato specificamente per lo sviluppo per la piattaforma Android.

DEFINIZIONE

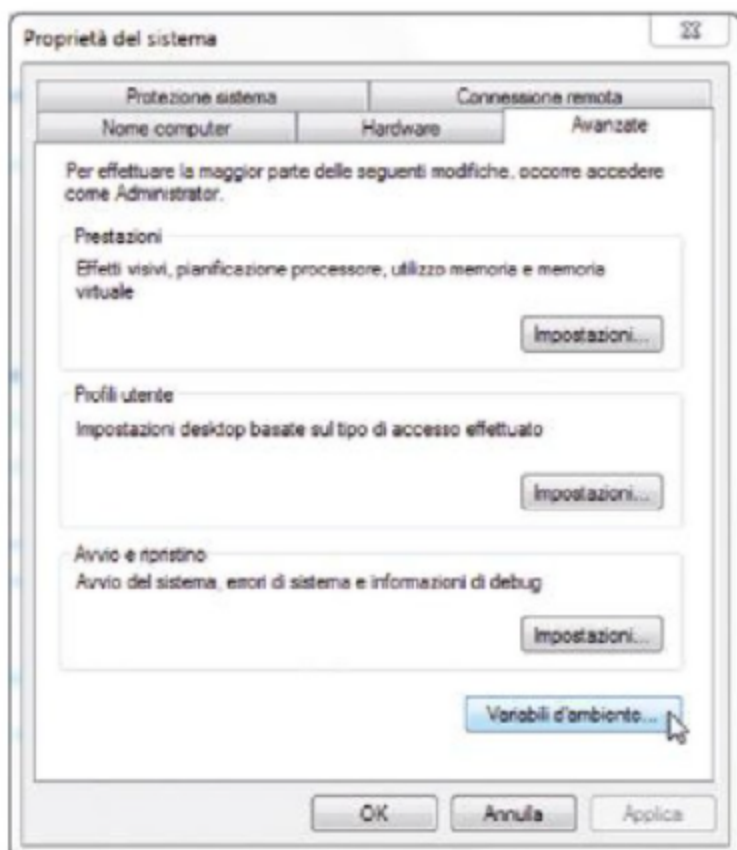
È l'acronimo di Integrated Development Environment, che significa ambiente di sviluppo integrato. Si tratta in sintesi di un software che aiuta i programmatori nello sviluppo del codice mettendo a disposizione del programmatore un editor di codice sorgente, un compilatore o un interprete, un tool di building automatico e un debugger.

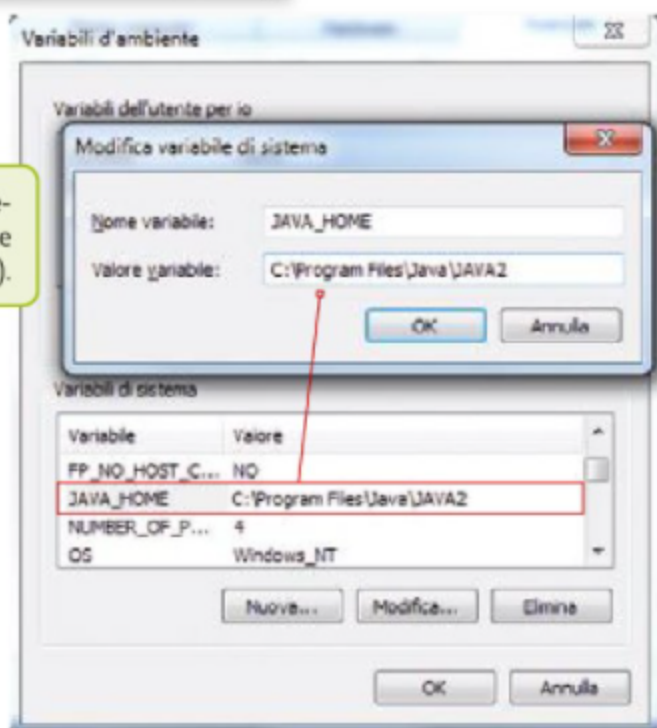
Prima di utilizzare l'ambiente di sviluppo Android Studio, dobbiamo installare le librerie e i compilatori Java (JDK – Java Development Kit).

Per installare la JDK dobbiamo collegarci al sito:

http://www.oracle.com/technetwork/java/javase/downloads/

Prima dobbiamo selezionare la versione di Java JDK e quindi il sistema operativo desiderato.

A volte può essere necessario dover inserire manualmente il percorso d'installazione della JDK, mediante le variabili d'ambiente per Windows: per fare questo apriamo il pannello di controllo, selezioniamo l'icona Sistema, quindi Impostazioni di sistema Avanzate, scheda Avanzate e infine Variabili d'ambiente.

Durante l'installazione conviene assegnare alla directory di JDK un nome facile da ricordare (ad esempio JAVA2).

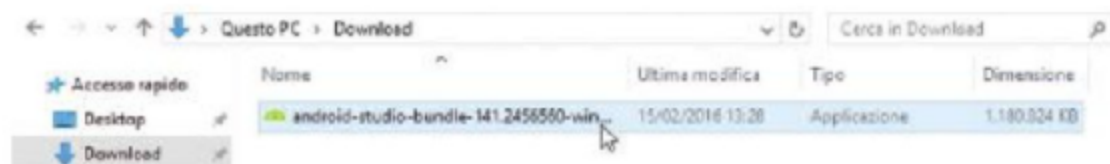
All'interno della sezione Variabili di sistema, aggiungiamo una nuova variabile con il nome JAVA_HOME e come percorso, il percorso nel quale abbiamo installato Java JDK (ad esempio C:\Program Files\Java \JAVA2): ▶

Durante l'installazione di Android Studio vengono anche installate le librerie **Android SDK**, necessarie per lo sviluppo di applicazioni native. Nonostante l'installazione avvenga in contemporanea con Android Studio le configurazioni vanno effettuate separatamente.

> **DEFINIZIONE**
>
> Android SDK è l'insieme delle librerie necessarie alla gestione dei dispositivi Android e comprende i componenti necessari per lo sviluppo degli applicativi, come ad esempio i driver.

Scaricare e installare Android Studio

La seguente procedura illustra come scaricare e installare Android Studio:

1. Prima di tutto colleghiamoci al sito dal quale scaricare Android Studio e le relative librerie: developer.android.com/sdk.
2. Selezioniamo il pacchetto per il sistema operativo che stiamo utilizzando, in questo caso Windows, quindi facciamo clic su Download Android Studio for Windows.

3. Dopo aver letto e confermato le condizioni di utilizzo proposte, facciamo clic sul pulsante Download Android Studio per iniziare a scaricare il pacchetto.
4. Una volta terminato il download ci posizioniamo nella cartella Download, a questo punto facciamo doppio clic sul file eseguibile, indicato nella finestra seguente, per iniziare la procedura di installazione.

5. Dopo aver fatto clic su Next nella finestra precedente (finestra di Welcome), dobbiamo scegliere quali componenti installare. Verranno installati automatica-

mente anche l'Android SDK e l'AVD Manager.

DEFINIZIONE

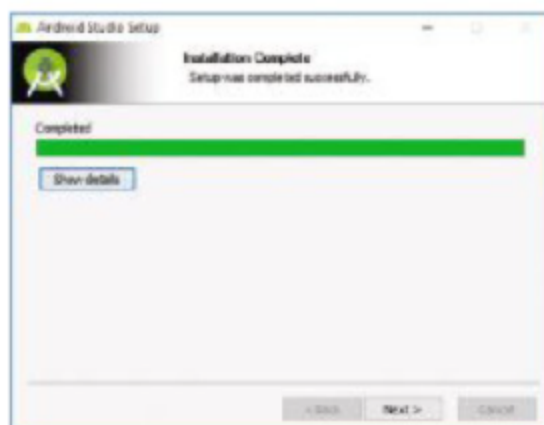
È un componente aggiuntivo che permette di creare e gestire degli emulatori Android. AVD è infatti l'acronimo di Android Virtual Device. Grazie agli AVD possiamo infatti testare e fare il debug di applicazioni Android senza bisogno di un dispositivo fisico.

Se abbiamo già installato sia Android SDK che AVD Manager, dobbiamo togliere il segno di spunta sulle caselle di scelta e passare all'inserimento manuale dei percorsi di installazione.

Nel nostro caso i componenti aggiuntivi non sono installati pertanto lasciamo le impostazioni suggerite dal programma e facciamo clic su Next per proseguire:

6. Dopo aver letto e accettato la licenza d'uso di Anrdoid SDK confermando con I Agree, dobbiamo scegliere i percorsi di installazione dei diversi componenti. Accettiamo il percorso proposto. Ricordiamo di annotare i percorsi di installazione per modifiche future: ▶

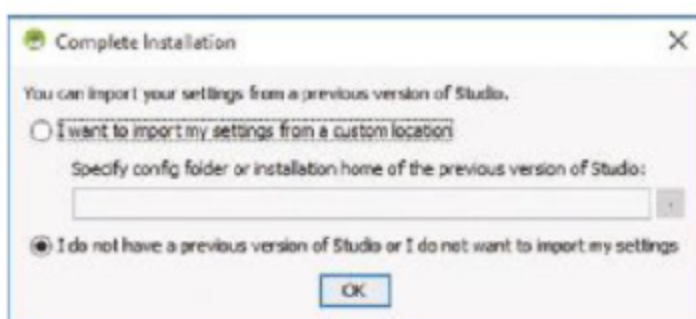
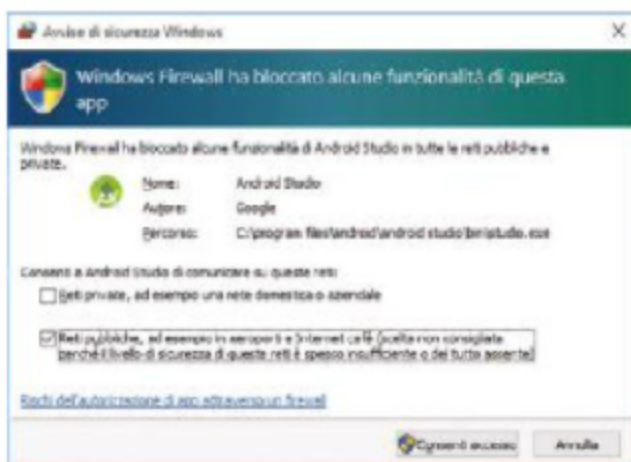
7. Decidiamo se creare un collegamento sul Desktop per avviare velocemente Android Studio, quindi facciamo clic su Install per avviare l'installazione vera e propria.

8. Una volta che l'installazione è stata completata facciamo clic su Next per continuare: ▶

9. Facciamo infine clic su Finish per avviare Android Studio: ▶

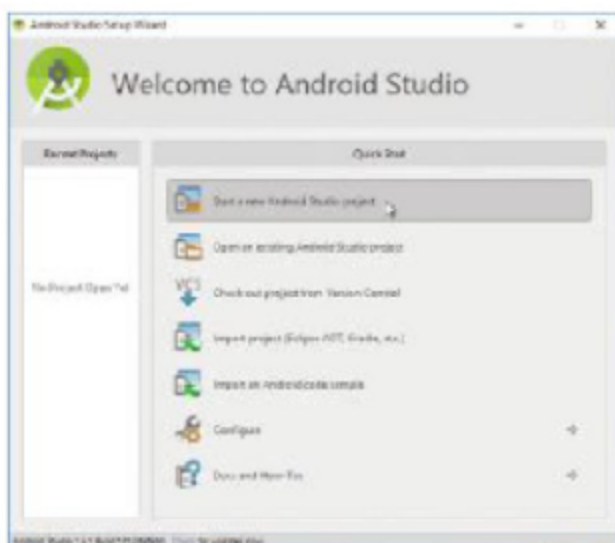
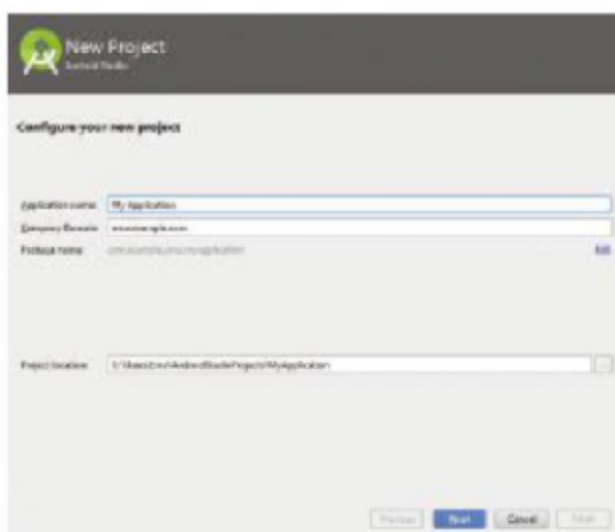
La procedura di installazione non è ancora terminata, mancano ancora alcuni file relativi all'SDK che Android Studio installerà solo durante la creazione del progetto relativo alla nostra prima applicazione.

Creare un'applicazione

La procedura illustra come creare la nostra prima applicazione, per terminare così l'installazione dei componenti aggiuntivi necessari all'uso di Android Studio.

1. Aprendo Android Studio ci viene richiesto di importare le impostazioni da una precedente versione di Android Studio. Siccome questa è la prima installazione selezioniamo la seconda opzione e facciamo clic su OK.
2. A questo punto dell'installazione viene richiesto se consentire l'accesso a Internet per far comunicare Android Studio e la rete attraverso il Firewall. Questo è necessario per poter utilizzare GRADLE, che verifica periodicamente la presenza di aggiornamenti necessari alle librerie di Android SDK.

DEFINIZIONE

È uno dei più avanzati strumenti di build automation del momento che permette di automatizzare in particolare la compilazione, la documentazione e il packaging dei programmi nonché di eseguire dei test automatici per verificare il comportamento dell'applicativo in diverse situazioni.

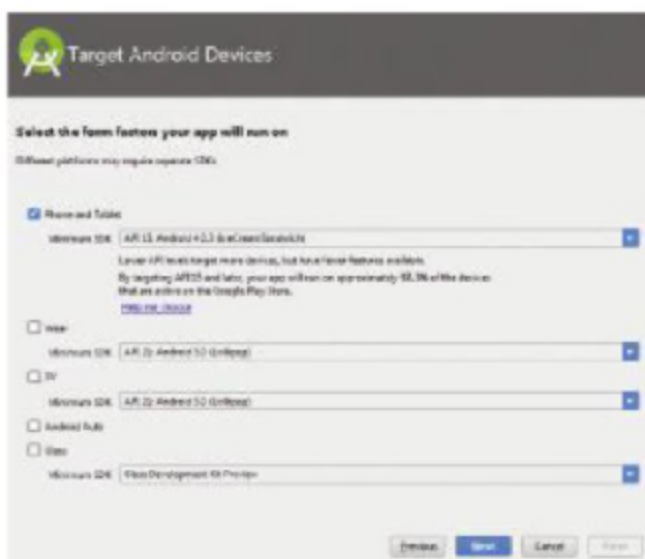
3. Appare ora la videata principale di Android Studio dalla quale possiamo creare un nuovo progetto, aprirne di esistenti, importare progetti creati in precedenza con altri pacchetti di sviluppo (ad esempio Eclipse) e accedere a tutte le impostazioni disponibili. Procediamo con la creazione di un nuovo progetto selezionando la voce Start a new Android Studio project: ▶

4. A questo punto dobbiamo scegliere il nome e il percorso della nostra applicazione. Il nome che sceglieremo sarà lo stesso della cartella con la quale il progetto sarà identificato e riconosciuto da Android Studio: ▶

Facciamo attenzione a scegliere accuratamente il nome da assegnare al progetto: infatti non sarà più possibile modificarlo in seguito, se non con procedure molto elaborate.

5. Una volta definito il nome dell'applicativo e il percorso di salvataggio dobbiamo scegliere il tipo di applicazione che andremo a sviluppare. I tipi sono raggruppati in cinque categorie:
   - Phone and Tablet: è il tipo che ci interessa, riguarda in particolare lo sviluppo di applicazioni native per smartphone e tablet dotati di sistema operativo Android. Quando si seleziona questa opzione si deve anche indicare la versione minima di Android su cui girerà l'applicazione.

DEFINIZIONE

Definiamo applicazioni native le applicazioni che sono state sviluppate utilizzando strumenti per il controllo diretto dei dispositivi interessati e non utilizzando software di terze parti che permettono un notevole riciclo del codice a fronte di una scarsa qualità dello stesso.

   - Wear: permette di sviluppare applicativi per gli smartwatch dotati di sistema operativo Android Wear. Anche in questo caso è necessario indicare una versione minima di compatibilità.
   - TV, Android Auto, Glass: permettono di sviluppare applicativi per quella ristretta parte di dispositivi che utilizzano le rispettive versioni di Android.

Selezioniamo quindi Phone and Tablet, lasciamo la versione suggerita (4.0.3) che copre il 97,3% dei dispositivi in circolazione, quindi facciamo clic su Next.

6. Appare la finestra di creazione del primo progetto che installa contestualmente l'Android SDK:

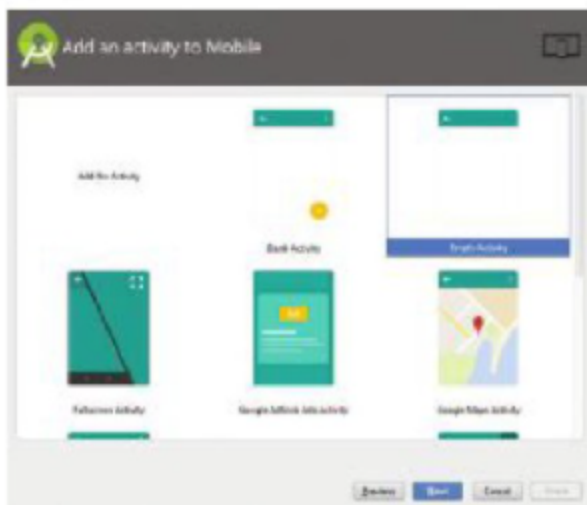
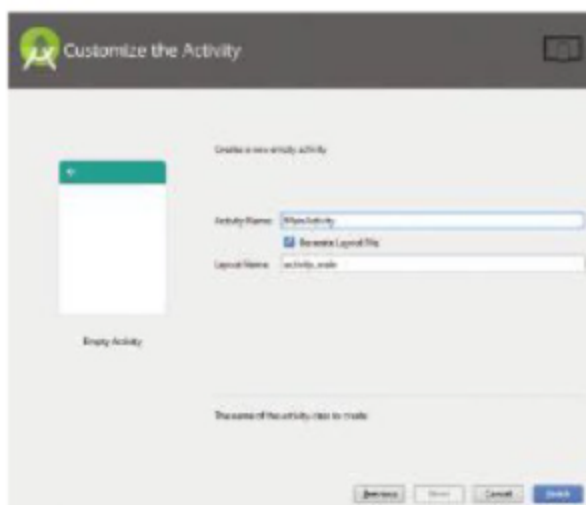
7. Terminata l'installazione ci viene mostrata una schermata di conferma di avvenuta installazione, della quale possiamo anche visualizzare i dettagli. Procediamo con la creazione della nostra prima applicazione facendo clic su Next.
8. Possiamo adesso selezionare il tipo di Activity che vogliamo inserire: ne esistono di diverse, ciascuna con una precisa funzione.

DEFINIZIONE

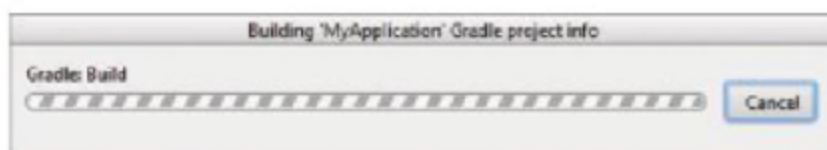
Activity è uno dei principali componenti di ogni applicazione Android e costituisce quella parte di applicazione che permette la comunicazione con l'utente. È infatti utilizzata per mostrare informazioni all'utente e comprende parte dei metodi per l'elaborazione delle informazioni stesse.

Vediamole in dettaglio:

- Blank Activity: quasi vuota, contiene un menu a tendina per le impostazioni e una piccola scorciatoia modificabile a nostro piacimento.
- Empty Activity: è la tipologia più semplice, non contiene nulla se non una TextView con scritto HelloWorld!
- Fullscreen Activity: è impostata per un funzionamento a schermo intero, rendendo quindi nascosta la barra delle notifiche.
- GoogleMaps Activity: integra un'istanza delle mappe di Google.
- Settings Activity: contiene dei componenti grafici molto utili per la regolazioni delle impostazioni dell'applicazione.
- NavigationDrawer Activity: integra un menu laterale molto utile per la navigazione tra varie activity.

Vi sono anche altre tipologie di activity preimpostate, ma esulano da quelli che sono i nostri obiettivi di apprendimento. Selezioniamo Empty Activity e facciamo clic su Next per procedere. ▶

9. Dobbiamo adesso scegliere il nome da assegnare alla activity e al relativo layout. Nel caso non volessimo generare automaticamente il file di layout associato all'activity possiamo togliere la spunta da Generate Layout File. In questo caso lasciamo i nomi proposti da Android Studio e facciamo clic su Finish per continuare. ▶

DEFINIZIONE

Il layout è il componente che definisce l'interfaccia grafica che verrà visualizzata dall'utente ed è associato a una o più activity. In sostanza il layout costituisce la parte visiva mentre l'activity rappresenta tutto il codice che ci sta sotto.

10. Appare la schermata che mostra la creazione del progetto con le specifiche definite in precedenza.

In Android Studio sono presenti moltissime scorciatoie che ci permettono di velocizzare la fase di programmazione. All'avvio del progetto appare una schermata denominata Tip of the day, che suggerisce qualche trucco utile.

11. Dopo aver letto qualche consiglio chiudiamo la schermata facendo clic su Close, finalmente avremo accesso a quella che è la pagina principale dalla quale possiamo gestire ogni aspetto e codice della nostra applicazione.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Comprendere il ruolo degli IDE	○	○	○
Riconoscere il ruolo di Android Studio	○	○	○
Capire il significato di GRADLE	○	○	○
Comprendere le tipologie di funzioni delle activity	○	○	○

Competenze	Scarso	Medio	Ottimo
Scaricare Java	○	○	○
Installare la JDK	○	○	○
Scaricare Android Studio	○	○	○
Installare Android Studio	○	○	○
Installare Android SDK e AVD Manager	○	○	○
Realizzare una applicazione di prova	○	○	○

3
LEZIONE

L'INTERFACCIA GRAFICA DI ANDROID STUDIO

In questa lezione impareremo

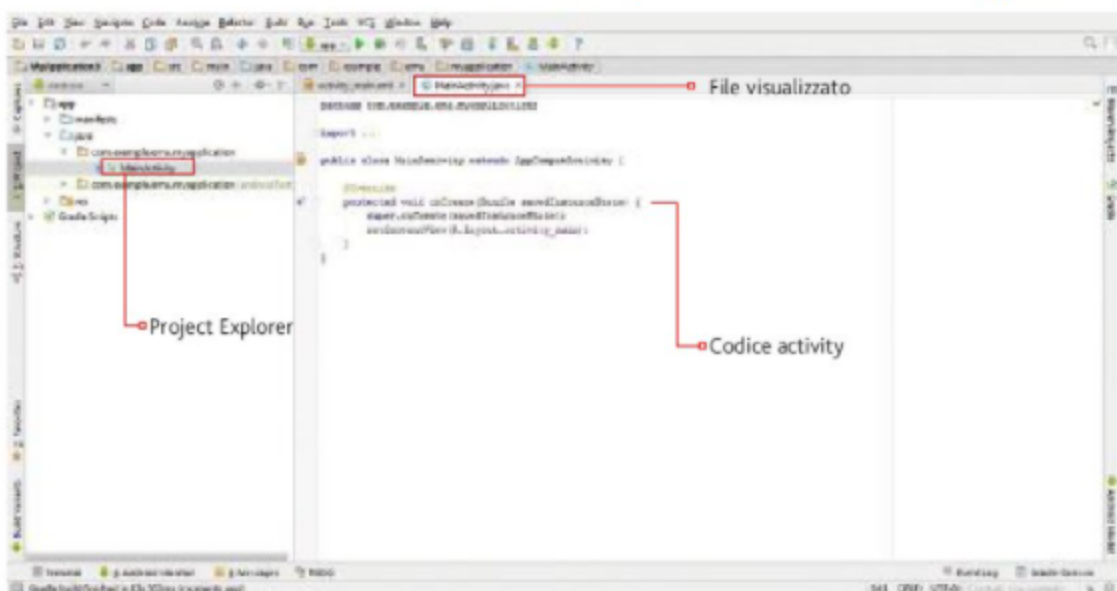
- a utilizzare l'ambiente di sviluppo Android Studio
- a far eseguire l'applicazione tramite emulatore
- a eseguire il debug dell'applicazione

L'ambiente di lavoro

Vediamo come è strutturata la finestra principale dell'ambiente di lavoro Android Studio: sulla colonna di sinistra vi è il Project Explorer, grazie al quale è possibile passare agevolmente da un file a un altro del progetto. Nella finestra centrale vi è l'editor che mostra, in questo caso, il codice dell'activity selezionata (MainActivity.java).

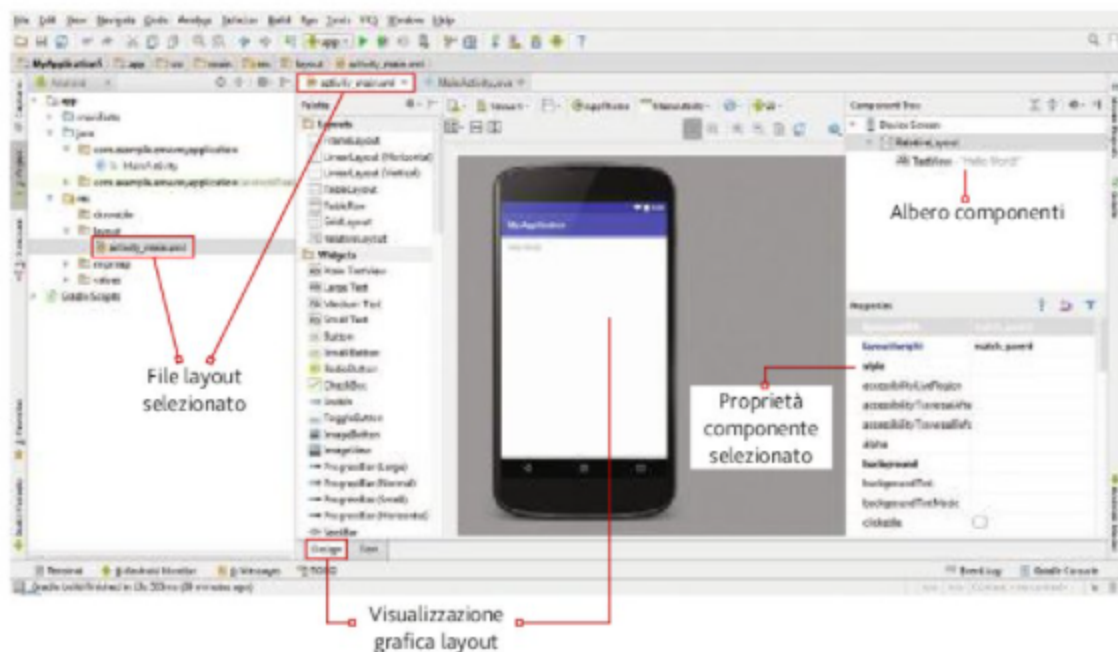
File visualizzato

Project Explorer

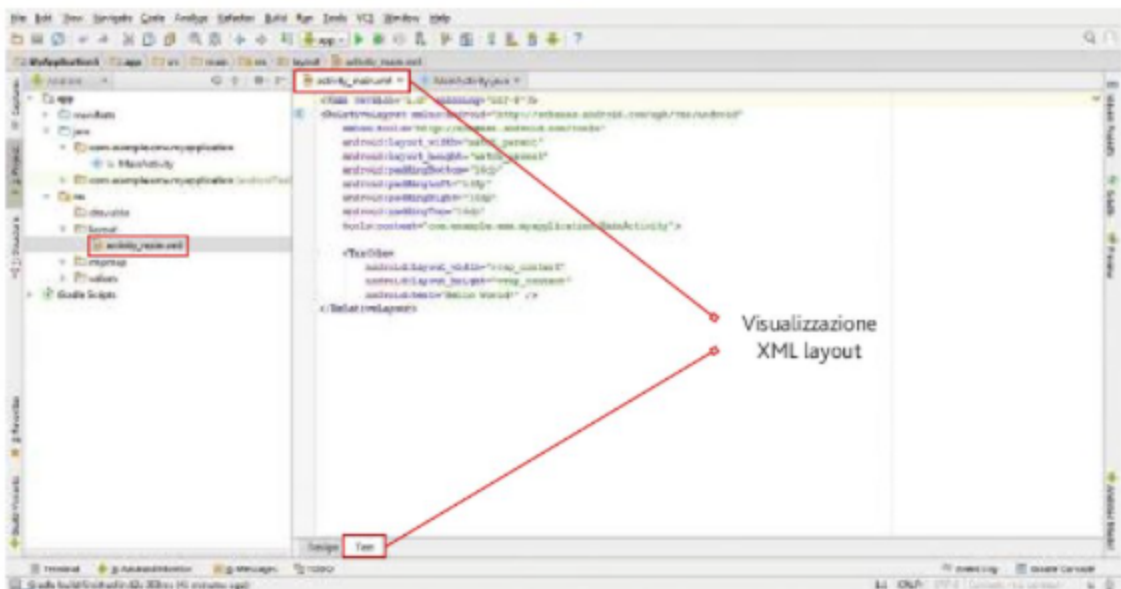
Codice activity

Utilizziamo il Project Explorer per visualizzare il file activity_main.xml, che descrive il layout del nostro applicativo. Android Studio ci propone subito un'interpretazione grafica del layout, molto utile qualora volessimo verificare l'aspetto della nostra applicazione senza avviarla su un dispositivo.

Notiamo sulla destra due riquadri importanti: Component Tree, che ci mostra tramite una struttura annidata i widget presenti all'interno del nostro layout, e Properties, che visualizza la lista delle proprietà relative al controllo selezionato.

DEFINIZIONE

I widget sono i controlli grafici che consentono l'interazione con l'utente, come ad esempio i pulsanti di azione, le caselle di testo o liste di opzioni. Il termine deriva dalla contrazione dei termini **wi**ndows e ga**dget**.

Cliccando su Text (accanto a Design) passiamo alla visualizzazione del layout in modalità testuale. In particolare il layout è definito in linguaggio XML e a volte può risultare molto più agevole modificare il layout utilizzando questa modalità.

DEFINIZIONE

XML (sigla di eXtensible Markup Language) è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Il Project Explorer

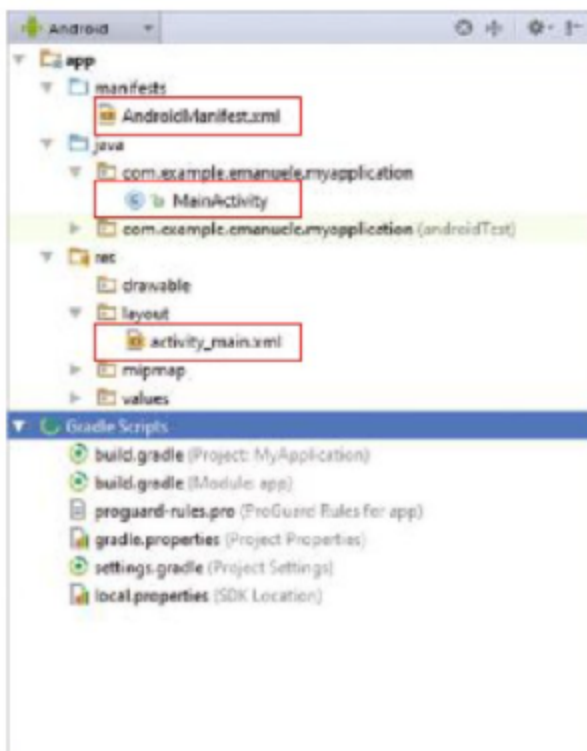
Il Project Explorer è uno strumento che ci permette di esplorare e navigare tra i file di cui è composto il nostro progetto. In particolare ci permette di accedere a tre file che sono la base del funzionamento della nostra applicazione:

- AndroidManifest.xml
- MainActivity.java
- activity_main.xml

AndroidManifest.xml

Quando creiamo un nuovo progetto, viene creato automaticamente il descrittore dell'applicazione, il file manifest chiamato AndroidManifest.xml.

DEFINIZIONE

I file manifest consentono di definire la struttura e i metadati XML dell'applicazione. Include un nodo per ogni componente (Attività, Servizi, Content Providers e Broadcast Receiver) e attraverso gli Intent Filter e i Permission determina come ogni componente interagisce con gli altri e le altre applicazioni.

Contiene inoltre il nome del package e altre informazioni, come ad esempio:

```
AndroidManifest.xml ×
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.emanuele.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Deve essere memorizzato nella cartella principale dell'applicazione e descrive i componenti dell'applicazione, in modo tale che il sistema operativo possa conoscere i componenti e le librerie usate dall'applicazione e necessarie per la sua corretta allocazione in memoria.

MainActivity.java

Come possiamo osservare pur creando un'applicazione vuota, Android Studio inserisce già delle porzioni di codice. In particolare effettua l'override del metodo onCreate, richiamando il costruttore della classe madre e settando il layout dell'activity con il metodo setContentView.

```
MainActivity.java ×
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

activity_main.xml

Contiene il codice XML che descrive il layout della activity. Fornisce informazioni riguardo al contenitore più esterno e ai widget in esso contenuti, in questo caso una TextView con scritto "Hello World!".

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.example.emanuele.myapplication.MainActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Vogliamo ora provare a eseguire la nostra prima applicazione. Per poterla provare abbiamo bisogno di un emulatore Android funzionante e/o di un dispositivo Android collegato in debug mode.

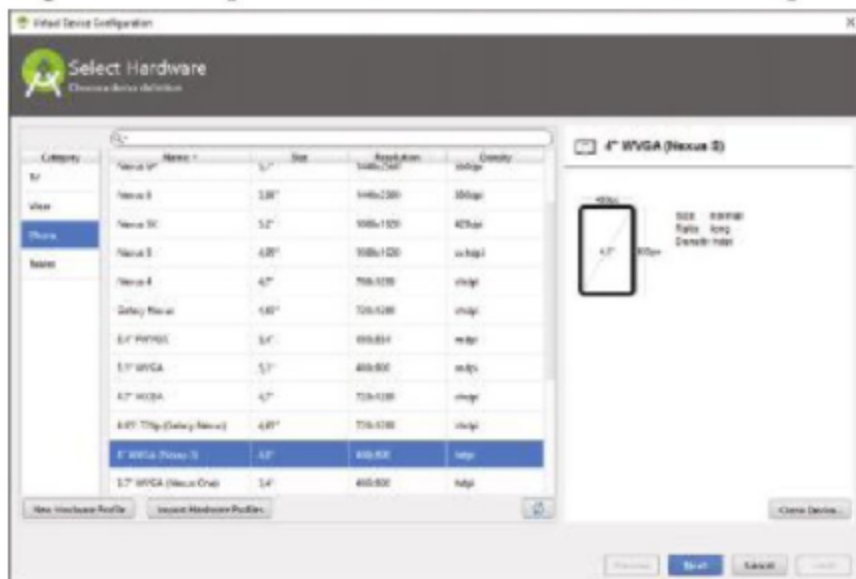
Creare un emulatore

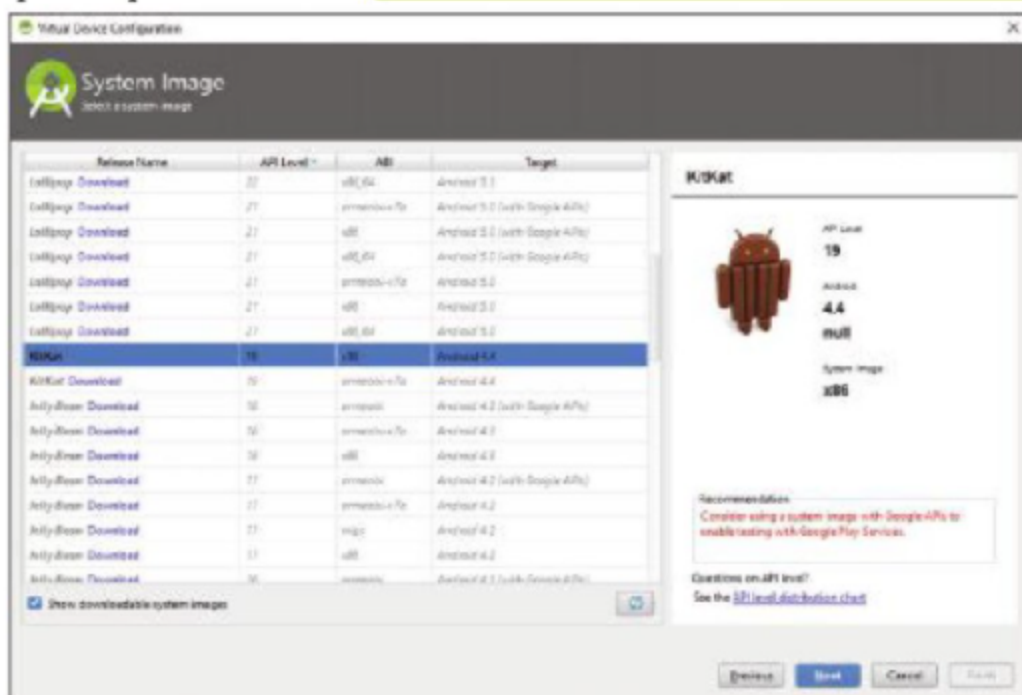
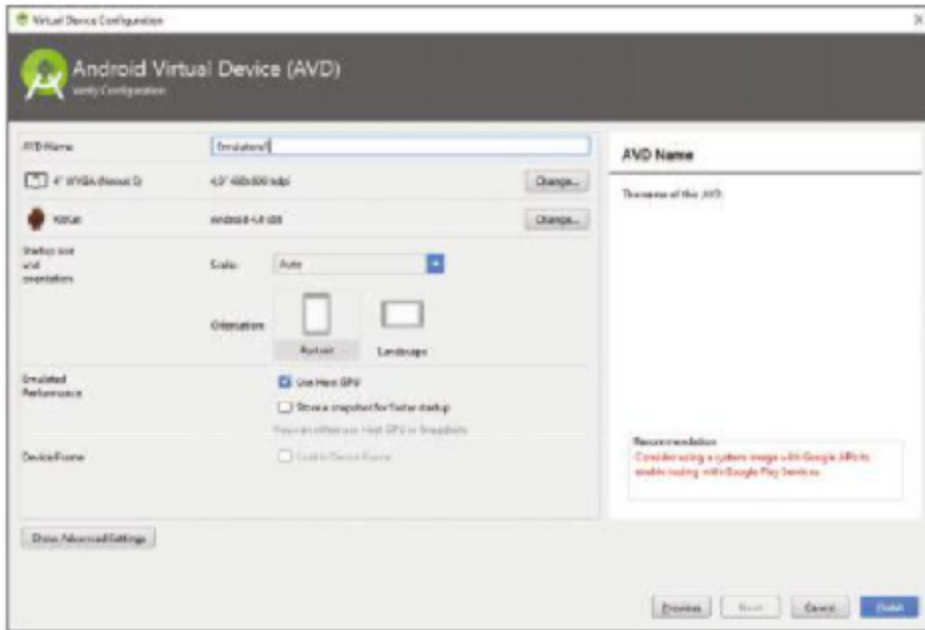
La procedura seguente mostra come configurare l'emulatore necessario per effettuare il test e il debug delle nostre applicazioni.

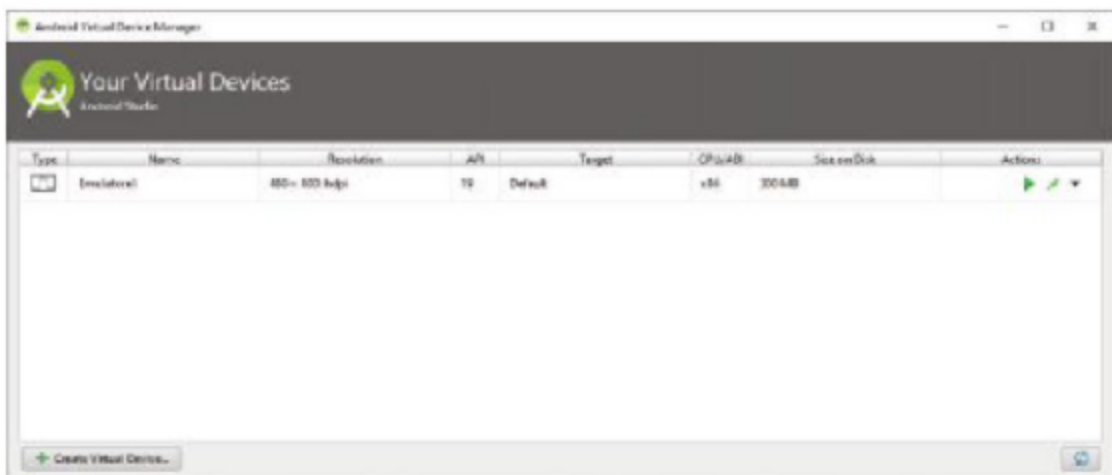
1. Dal menu Tools → Android selezioniamo AVD Manager.
2. Siccome non è presente alcun emulatore procediamo creandone uno nuovo facendo clic su Create Virtual Device:

3. La seguente finestra consente di selezionare la risoluzione in pixel dell'emulatore.

Scegliere una risoluzione troppo alta può portare a dei rallentamenti nell'esecuzione dell'emulatore, nonché a un aumento della memoria RAM necessaria al PC per il funzionamento dell'emulatore stesso.

In questo caso scegliamo un dispositivo mobile Nexus 4 con risoluzione pari a 480×800:

4. Selezioniamo ora la versione di sistema operativo Android che dovrà essere eseguita dall'emulatore, sceglicdola tra quelle disponibili.

La scelta del sistema operativo influisce molto sulle prestazioni dell'emulatore, pertanto consigliamo di utilizzare una versione non troppo recente per non appesantire l'elaborazione del processore (ad esempio la versione x86).

5. Scegliamo infine il nome da assegnare all'emulatore (in questo caso Emulatore1) e l'orientamento dello schermo all'avvio: orizzontale (Portrait) o verticale (Landscape):

6. Viene proposto ora l'elenco degli emulatori creati finora. Scegliamo l'emulatore che desideriamo mandare in esecuzione e quindi facciamo clic sul pulsante Play ▸ della colonna Actions, per avviarlo:

Creare più emulatori con differenti risoluzioni e versioni di Android, può essere utile per verificare come si comporta la nostra applicazione su dispositivi con caratteristiche fisiche e software diverse.

7. A questo punto viene mostrato l'emulatore in esecuzione: è pronto per testare le nostre applicazioni. Oltre alla schermata principale, nella quale possiamo utilizzare il mouse per simulare il tocco (tap) dello schermo (chiamato tap) viene mostrata una colonna, collocata sul lato destro del dispositivo, che emula i pulsanti fisici presenti sul dispositivo.

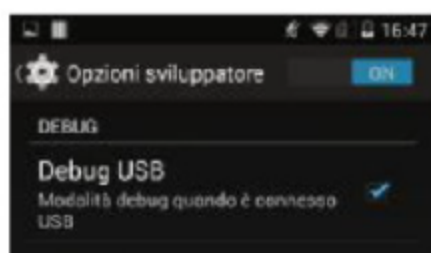
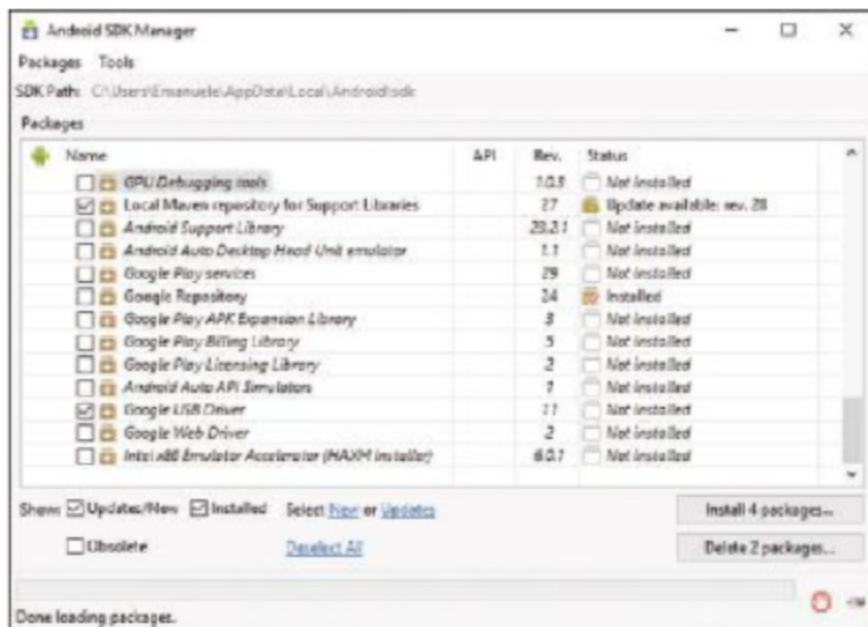
Il prossimo paragrafo spiega come configurare un dispositivo Android per utilizzarlo per il debug al posto dell'emulatore.

Configurazione dispositivo fisico

Per effettuare il debug delle applicazioni possiamo utilizzare il dispositivo fisico, cioè lo smartphone o il tablet vero e proprio dotato di sistema operativo Android. Per fare questo dobbiamo tuttavia installare i driver necessari tramite la procedura seguente.

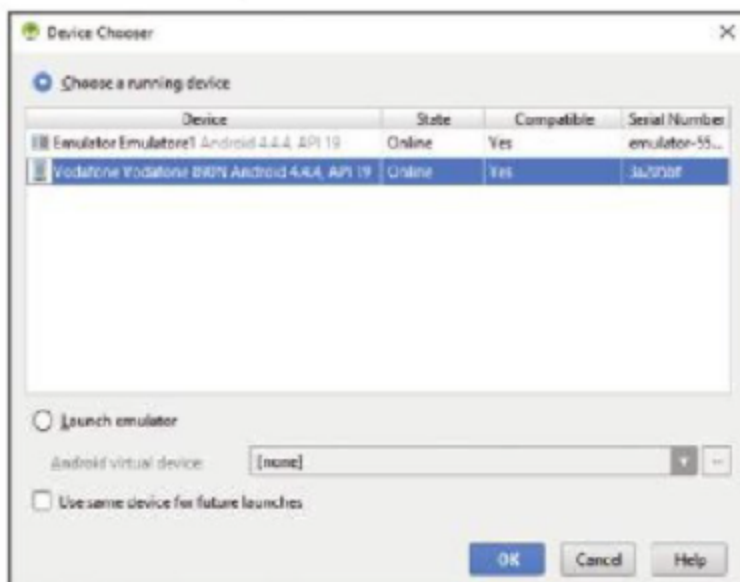
1. Posizioniamoci nella cartella C:\Users\NomeUtente\AppData\Local\Android\sdk e avviamo l'eseguibile SDK Manager.

Se volessimo invece utilizzare un dispositivo mobile dotato di sistema operativo Windows, sarà necessario prima di tutto installare dei particolari driver (USB drivers) che lo possano rendere compatibile con Android Studio. Il seguente link mostra alcuni driver disponibili: https://developer.android.com/studio/run/oem-usb.html.

2. Selezioniamo gli aggiornamenti disponibili e Google USB Driver e procediamo con l'installazione.

3. Ora possiamo collegare il nostro dispoditivo Android con il cavo USB. Per poter utilizzare il dispositivo per il debug bisogna abilitare le Impostazioni sviluppatore dal menu impostazioni e attivare la funzione Abilita Debug USB.

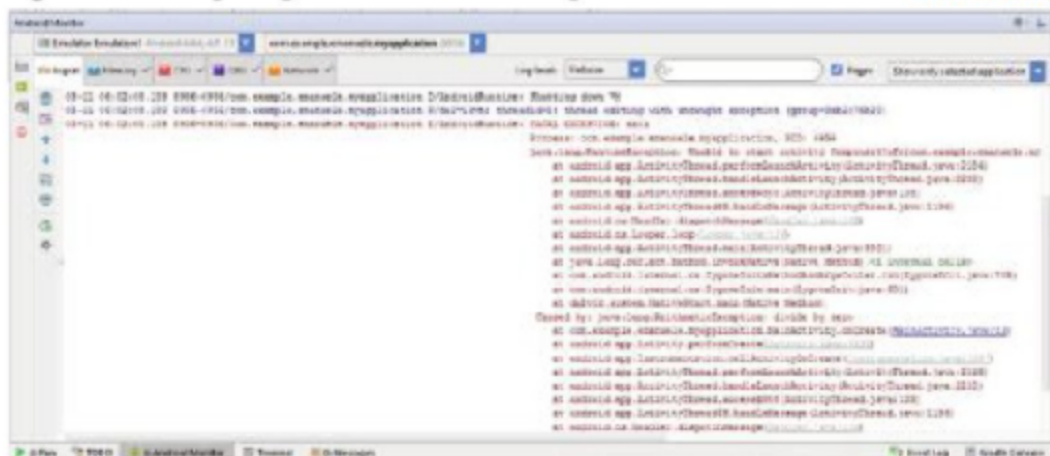
Ogni volta che colleghiamo il dipositivo al PC ci verrà chiesta un'ulteriore autorizzazione per il debug. ▶

Mandare in esecuzione un'app

1. Per avviare la nostra applicazione selezioniamo Run app dal menu Run, otteniamo una schermata simile a questa:

Il numero di voci visualizzate dipende da quanti emulatori abbiamo avviato e se vi sono o meno dispositivi fisici collegati. In questo caso abbiamo un emulatore avviato e un dispositivo fisico collegato tramite USB. Selezioniamo il dispositivo (Device) sul quale si desidera avviare l'applicazione e confermiamo con OK.

2. Sullo schermo del nostro emulatore/dispositivo vediamo adesso l'applicazione in esecuzione, in questo caso si tratta solo di una schermata bianca con la scritta HelloWorld! ▶

Una volta avviata l'applicazione è disponibile un nuovo strumento in Android studio: si tratta di Android monitor, serve per verificare le attività svolte dall'applicazione, vediamone le cinque principali:

- Logcat: è uno strumento di log, raccoglie tutti i log che l'applicazione invia tramite appositi comandi. Tiene traccia delle funzioni che sono state eseguite e di quelle in esecuzione. Mostra inoltre eventuali errori di runtime, come mostrato dalla finestra seguente, nella quale possiamo notare un tipico errore causato da una division by 0:

- Memory: mostra un grafico che rappresenta l'utilizzo della memoria RAM da parte dell'applicazione nel corso del tempo.

- CPU, GPU e Network mostrano alcuni grafici, analogamente rispetto a Memory, legati all'utilizzo rispettivamente del processore, della scheda grafica e della rete Internet.

Nella colonna di sinistra appaiono alcuni pulsanti relativi a funzioni poco utilizzate, in particolare i primi due permettono di acquisire uno screenshot dello schermo dell'emulatore o dispositivo collegato e di registrare un video di funzionamento dell'applicazione.

Effettuare il debug con Android Studio

Il programmatore, durante l'attività di scrittura di un programma, deve tener conto dei possibili errori che si producono. Tali errori possono essere raggruppati in tre categorie:

- errori di compilazione causati dall'utilizzo di parole che non appartengono al linguaggio oppure dalla costruzione non corretta di istruzioni del codice;

- errori in fase di esecuzione, chiamati anche errori di run time, segnalati durante l'esecuzione del programma;
- errori logici che generano risultati diversi da quelli attesi.

Per evitare di commettere errori, il programma deve essere progettato tenendo conto di tutti i possibili valori che l'utente potrà immettere durante l'esecuzione.

L'ambiente Android Studio mette a disposizione uno strumento che consente di individuare i diversi tipi di errore e di apportare al codice le opportune correzioni, chiamato debugger. L'attività di individuazione e correzione degli errori del codice sorgente viene comunemente chiamata fase di debugging.

DEFINIZIONE

Il termine debugger ha una origine molto lontana nel tempo, venivano infatti chiamati così i lavoratori che avevano il compito di ripulire le valvole dei primi computer dai nidi di alcuni tipi di coccinelle (dall'inglese bug che significa appunto coccinella). Più recentemente il termine ha preso via via un significato diverso, attualmente indica un software che ripulisce il programma dagli errori.

DEFINIZIONE

L'attività di debugging consiste nel rilevare ed eliminare gli errori di programmazione, in tale fase ci occupiamo innanzitutto degli errori sintattici, che impediscono di mandare in esecuzione un programma.

Un tipico errore è rappresentato dalla mancanza del punto e virgola al termine dello statement.

Tuttavia non è detto che il compilatore rilevi tutti gli errori, soprattutto se sono di carattere logico. In generale, quando gli errori sono più di uno, conviene cercare di risolvere il primo della lista e procedere così via via fino all'ultimo. Inoltre molto spesso alcuni errori si propagano, quindi risolvendo il primo otteniamo una lista assai più ridotta. Non conviene, perciò, eseguire sempre alla lettera le indicazioni del compilatore, ma partire da tali indicazioni ed esaminare il codice per capire l'azione che dobbiamo intraprendere.

Per utilizzare il debugger di Android Studio dobbiamo applicare tre concetti basilari:
- l'impostazione dei punti di arresto (breakpoint);
- l'ispezione del contenuto delle variabili;
- l'esecuzione del programma step by step (una riga alla volta).

I punti di arresto (breakpoints)

Per impostare un breakpoint dobbiamo fare doppio clic accanto alla riga di codice desiderata: apparirà un pallino rosso.

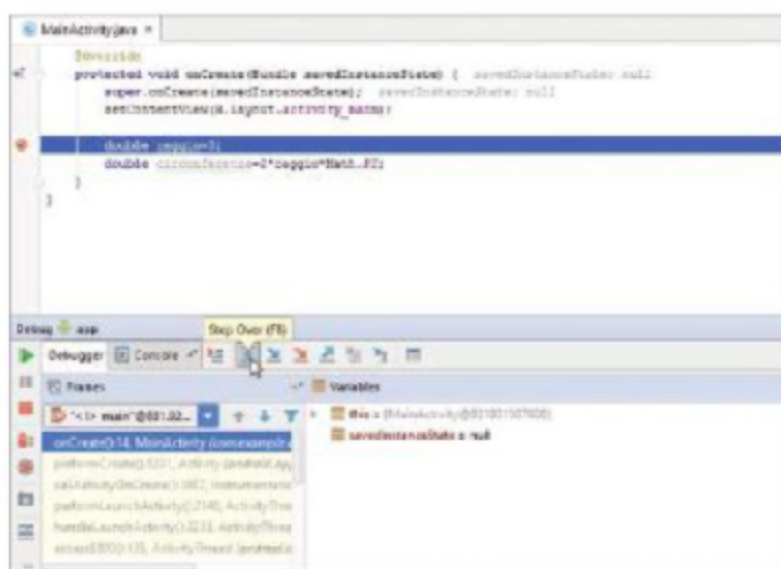
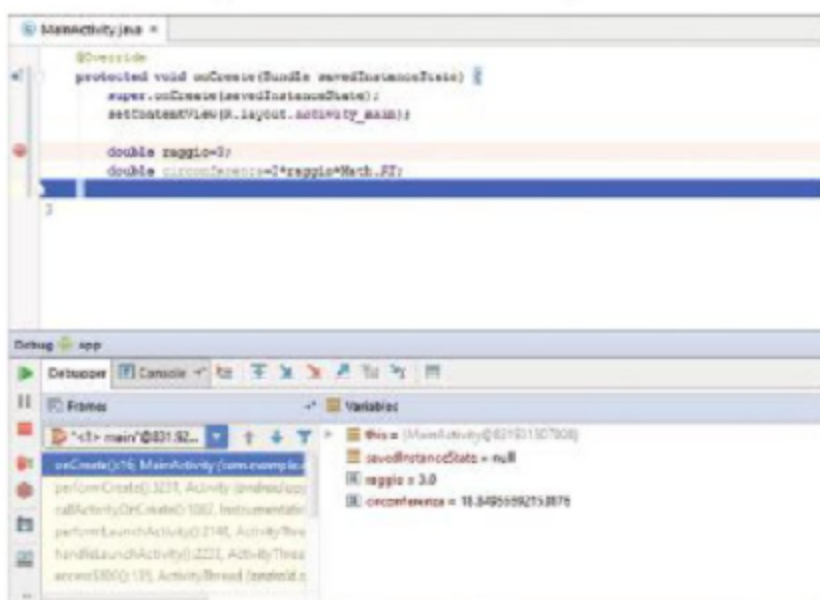
DEFINIZIONE

Un breakpoint è sostanzialmente uno strumento che consente di eseguire un programma con la possibilità di interromperlo quando si verificano determinate condizioni, allo scopo di acquisire informazioni su un programma in esecuzione. Per impostare un breakpoint dobbiamo fare doppio clic accanto alla riga di codice desiderata: apparirà un pallino rosso.

Per poter facilmente verificare il funzionamento del debug dobbiamo aprire il file MainActivity.java e modificarlo come segue. Notiamo l'aggiunta di un breakpoint, necessario per il test della funzionalità.

Con un clic del pulsante destro sopra il pallino di un breakpoint possiamo eventualmente impostare i breakpoint condizionali, utili a interrompere l'esecuzione del programma soltanto in caso di condizione verificata:

```
MainActivity.java
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        double raggio=3;
        double circonferenza=2*raggio*Math.PI;
    }
}
```

Per avviare il debug selezioniamo la voce Debug 'app' dal menu Run, selezioniamo poi il dispositivo sul quale effettuare il debug, e attendiamo l'inizializzazione della schermata Debugger.

Una volta avviato il debugger possiamo notare nella sezione Variables i contenuti delle variabili man mano che il programma viene eseguito. Per riprendere l'esecuzione del programma possiamo utilizzare il rettangolino giallo affiancato dal triangolo verde (Resume) che riprende l'esecuzione normale del programma fino al prossimo breakpoint impostato o alla fine del programma. Il quadratino rosso (Terminate) termina l'esecuzione del programma. Possiamo anche eseguire il programma in modalità passo passo (step by step); per fare questo è possibile utilizzare una funzione chiamata Step Into che esegue le istruzioni successive, una alla volta, entrando anche nei metodi, se richiamati nel proseguo del programma, oppure è possibile usare la funzione Step Over, che esegue le istruzioni successive, una per volta, senza tuttavia eseguirle all'interno dei metodi.

Dopo aver eseguito le due istruzioni successive nel programma visto sopra, possiamo osservare il valore calcolato e memorizzato all'interno della variabile circonferenza.

Toast

Abbiamo visto come effettuare il calcolo e l'assegnazione a una variabile, tuttavia senza il debug non avremmo mai potuto ottenerne il risultato a video. Esiste un modo assai semplice per comunicare dei dati in output sul display del dispositivo mobile, si tratta della classe Toast, che mette a disposizione dei messaggi a scomparsa automatica chiamati appunto toast.

DEFINIZIONE

Il nome deriva dal fatto che così come il toast viene catapultato fuori dal tostapane al termine della cottura, anche il messaggio di testo viene visualizzato per alcuni istanti sullo schermo, per poi scomparire. La forma più usata prevede di impostare i parametri durata e testo del messaggio, senza indicazioni sulla posizione in cui verrà visualizzato, che in genere è in basso e al centro dello schermo.

Proviamo a modificare il codice dell'esempio precedente come segue:

```
MainActivity.java ×
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        double raggio=3;
        double circonferenza=2*raggio*Math.PI;
        Toast toast = Toast.makeText(this,"La circonferenza è " + circonferenza,Toast.LENGTH_LONG);
        toast.show();
    }
}
```

Notiamo che il metodo makeText della classe Toast vuole in ingresso tre parametri:

1. il contesto al quale il Toast va associato (in questo caso this);
2. il testo da visualizzare nel messaggio;
3. la durata del messaggio che può assumere due valori che corrispondono a due costanti presenti nella classe Toast: LENGTH_SHORT per una breve durata e LENGTH_LONG per una durata maggiore.

Uno dei tipici errori che viene commesso è quello di non invocare il metodo show(), senza di questo metodo nessun messaggio viene visualizzato: sostanzialmente è come se il toast restasse dentro al tostapane, con il rischio di bruciarsi!

Testiamo adesso l'applicazione sull'emulatore, notiamo che all'apertura dell'applicazione viene mostrato il messaggio che comunica il valore della circonferenza, sparendo dopo pochi secondi.

Prova adesso!

- Testare una applicazione sul dispositivo mobile
- Utilizzare il debugger
- Utilizzare la classe Toast

Crea un'applicazione Primi

Crea una applicazione e testala sul tuo dispositivo mobile (in caso tu non l'avessi a disposizione utilizza l'emulatore di Android Studio e armati di pazienza!), in grado di generare 10 messaggi brevi di tipo Toast che visualizzino i primi 10 numeri primi.

Esercitiamoci

Test Vero/Falso

Indica, barrando la relativa casella, se le seguenti affermazioni sono vere o false.

1. Il Component Tree mostra gli errori di compilazione. V F
2. Il riquadro Properties mostra le proprietà relative al controllo selezionato. V F
3. Il layout è definito in linguaggio HTML. V F
4. XML è un linguaggio per la definizione di markup. V F
5. Il Project Explorer ci consente di vedere lo stato delle variabili durante l'esecuzione. V F
6. Il file manifest descrive la struttura e i metadati dell'applicazione. V F
7. Il file manifest è un file java. V F
8. La versione del sistema operativo dell'emulatore non influenza le prestazioni. V F
9. Per il debug delle applicazioni possiamo utilizzare solo emulatori. V F
10. I Google USB Driver servono per poter installare emulatori. V F
11. Per utilizzare un dispositivo per il debug basta collegarlo con il cavo USB. V F
12. Per poter attivare il debug USB dobbiamo abilitare le opzioni sviluppatore. V F
13. Possiamo sempre scegliere su quale dispositivo eseguire l'applicazione. V F
14. Logcat mostra la quantità di RAM utilizzata dall'applicazione. V F
15. Android Studio integra uno strumento di debugging. V F
16. È possibile inserire un breakpoint all'interno del codice. V F
17. L'esecuzione step by step non consente l'ispezione della variabili. V F
18. I Toast vengono mostrati solo se l'applicazione è in esecuzione su emulatore. V F

Domande a risposta multipla

Indica la risposta corretta barrando la casella relativa.

1 Quale tipo di errori sono causati dall'utilizzo di parole che non appartengono al linguaggio?

a. errori in fase di esecuzione
b. errori di allocazione della memoria
c. errori di compilazione
d. errori logici

2 Qual è il nome della console di debug di Android Studio?

a. GRADLE
b. Logcat
c. LogMe
d. Toast

3 Quali delle seguenti costanti possono essere utilizzate per indicare la durata di un Toast?

a. LENGTH_LONG
b. LONG_LENGTH
c. SHORT_LENGTH
d. LENGTH_SHORT

4 Le Permission all'interno del file manager descrivono:

a. quali utenti possono usare l'applicazione
b. come ogni componente interagisce con gli altri
c. quali tipi di input l'applicazione supporta
d. quanta memoria l'applicazione occupa su disco

Esercitiamoci

5 L'AVD consente di:

a. creare progetti Android
b. utilizzare le classi dei package
c. utilizzare le librerie SDK per Android
d. utilizzare l'emulatore di dispositivo mobile

6 L'area di console presenta:

a. i progetti, le classi e i package
b. l'input e l'output del programma
c. i breakpoints del programma
d. le variabili locali del programma

7 L'area di package explorer presenta:

a. i progetti, le classi e i package
b. l'input e l'output del programma
c. i breakpoints del programma
d. le variabili locali del programma

8 Gli errori di programmazione possono essere sintetizzati in 3 tipologie, quali?

a. errori logici, semantici e sintattici
b. errori di compilazione, logici e semantici
c. errori di compilazione, logici e di esecuzione
d. errori di esecuzione, logici e di run time

9 Per verificare il contenuto delle variabili dobbiamo:

a. aggiungere un breakpoint
b. attivare la finestra Variabili
c. eseguire il programma step by step
d. compilare la classe

Problemi

Progetta e realizza completamente il codice in Java per Android che risolva il problema proposto.

1 Crea un progetto che mostri una serie di messaggi di tipo toast indicanti i nomi dei sette nani (Eolo, Mammolo, Cucciolo, Brontolo, Pisolo, Dotto, Gongolo).

2 Crea un progetto che mostri un messaggio di tipo toast indicante il numero pi greco: verifica quanti decimali al massimo si possono visualizzare.

3 Crea un progetto che mostri una serie di messaggi di tipo toast che mostri dieci numeri casuali, ciascuno compreso tra 1 e 100.

4 Crea un progetto che mostri una serie di messaggi di tipo toast indicanti i primi 4 numeri perfetti: un numero si dice perfetto quando è pari alla somma dei suoi divisori.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Riconoscere gli elementi dell'ambiente di lavoro Android Studio	○	○	○
Capire il significato di Virtual Device	○	○	○
Comprendere la differenza tra emulatore Android e dispositivo fisico	○	○	○
Individuare i driver necessari al collegamento di un dispositivo ad Android Studio	○	○	○
Riconoscere il ruolo di Android Monitor	○	○	○
Comprender il ruolo del debug Android	○	○	○
Riconoscere la differenza in debug tra Step Into e Step By Step	○	○	○

Competenze	Scarso	Medio	Ottimo
Effettuare il debug di una applicazione con emulatore Android	○	○	○
Testare il debug di una applicazione collegando il dispositivo mediante USB	○	○	○
Saper eseguire una applicazione	○	○	○
Utilizzare Android Monitor	○	○	○
Saper collocare breakpoint	○	○	○
Mostrare a video messaggi a tempo (toast)	○	○	○

4

LEZIONE

I WIDGET

In questa lezione impareremo

- a riconoscere il ruolo del layout
- a gestire i controlli dell'interfaccia grafica
- ad associare eventi tramite Listener

La modifica del layout

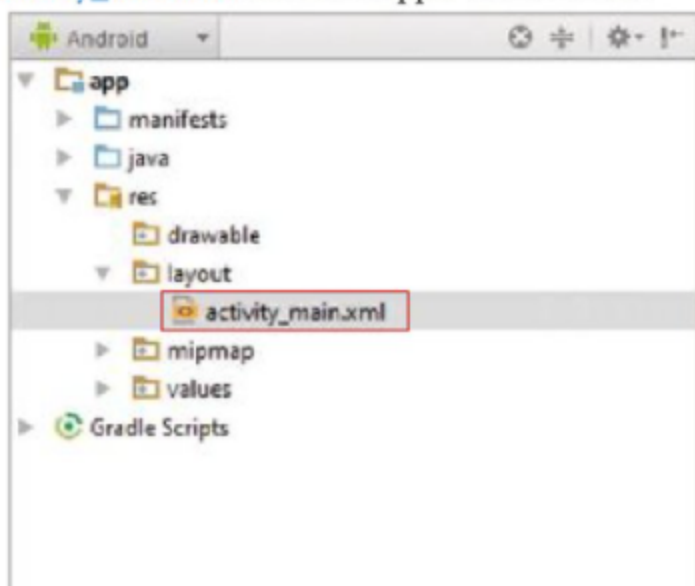
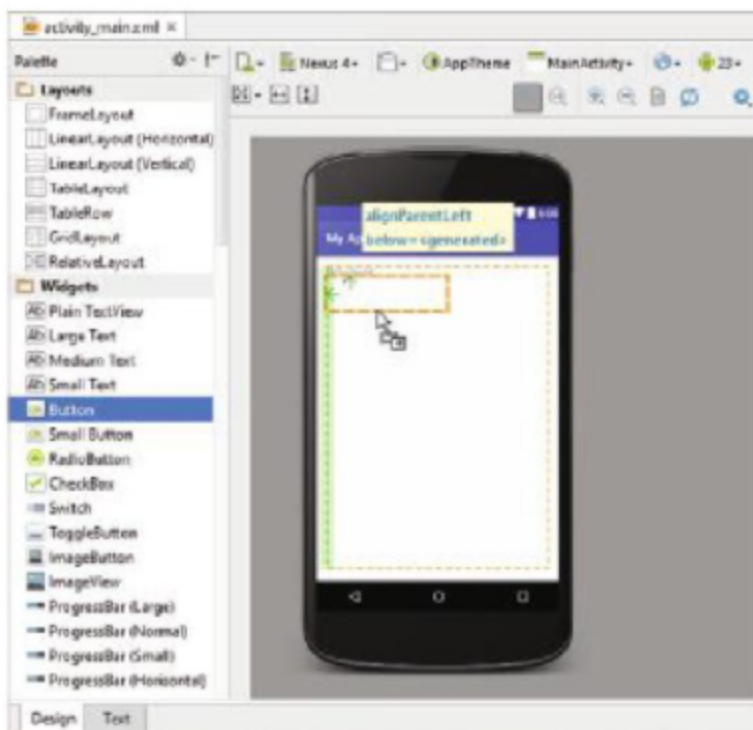
Per far comunicare l'utente con il dispositivo mobile, Android mette a disposizione moltissimi widget utilizzabili dal programmatore nelle proprie app. Alcuni controlli grafici, come ad esempio le TextView, hanno il solo scopo di trasmettere informazioni all'utente, mentre altre, come le EditView, vengono utilizzate per poter ricevere informazioni che l'utente digita tramite tastiera. La maggior parte dei widget in realtà può essere utilizzata in entrambi i modi, modificandone le proprietà per comunicare con l'utente e intercettando determinati eventi a scopo decisionale.
Per poter utilizzare questi controlli all'interno della nostra applicazione dobbiamo modificare il file activity_main.xml, che come già accennato descrive l'aspetto visivo che la nostra applicazione assume quando viene eseguita.

Per poter editare il file e creare componenti e schermate GUI possiamo procedere in due modi:

- trascinando i componenti grafici (Drag & Drop);
- modificando il codice sorgente del file .xml.

Trascinare i componenti grafici

Android Studio è dotato di uno strumento grafico che ci permette di aggiungere controlli alla nostra interfaccia semplicemente trascinandoli all'interno dell'anteprima che ci mostra.

Utilizzando il Project Explorer ci spostiamo all'interno della cartella res/layout e apriamo il file activity_main.xml facendo doppio clic su di esso.

Ci viene quindi mostrata l'anteprima del layout. Selezioniamo un widget (in questo caso un semplice bottone), lo trasciniamo sull'anteprima dello smartphone, quindi rilasciamo il pulsante sinistro del mouse per aggiungerlo al nostro layout nella posizione corrente, come mostrato nella seguente figura:

Modificare il file XML

Quella che Android Studio ci propone come anteprima non è altro che un rendering grafico del file activity_main.xml che, come suggerisce l'estensione, contiene una descrizione del layout in formato XML. Possiamo individuare la parte di codice relativa al pulsante che abbiamo appena inserito dal tag:

```
<Button … />
```

All'interno del tag ci sono alcune proprietà che sono state già assegnate da Android Studio, come possiamo notare ciascuna proprietà viene identificata con la seguente riga di codice:

```
android:nomeProprietà=valoreProprietà
```

activity_main.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.example.emanuele.myapplication.MainActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

Codice XML relativo al pulsante

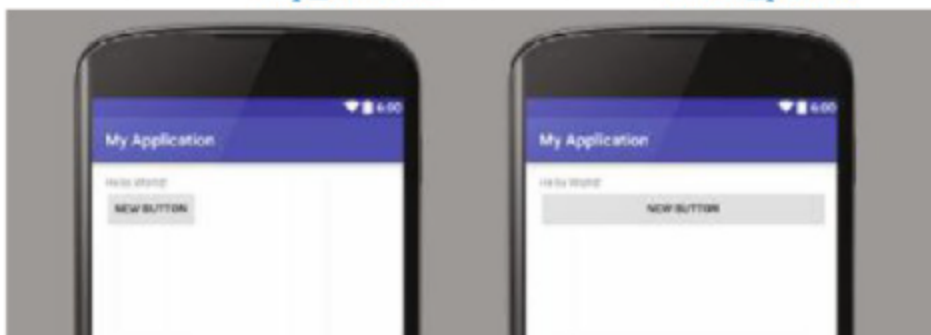
In realtà alcune proprietà dei widget possono essere osservate anche nell'ambiente grafico del layout. Per modificare le proprietà del componente grafico attraverso il layout dobbiamo rientrare in modalità grafica: dopo aver selezionato col mouse un controllo, osserviamo la scheda Properties, appaiono due colonne, in quella di sinistra appare la proprietà e in quella di destra il valore a essa associato:

Tra le principali proprietà comuni a tutti i widget troviamo:

- id: è il nome con il quale il controllo viene univocamente identificato. Utilizzato principalmente per recuperare il riferimento al controllo da codice.
- Layout:width indica la larghezza del controllo, oltre a valori in pixel (px) e density pixel (dp) può assumere i valori wrap_content e match_parent:
  - Wrap_content: il controllo si estende in larghezza tanto quanto basta affinché il suo contenuto sia ben visibile (in pratica se il testo all'interno del bottone fosse più lungo il bottone si allungherebbe fino a mostrarlo completamente).
  - Match_parent: il controllo si estende in larghezza fino al suo contenitore padre.

DEFINIZIONE

Dp: acronimo di Density-independent Pixel. È un'unità di misura astratta basata sulla densità fisica di pixel dello schermo. Un dp equivale a un pixel su uno schermo con densità pari a 160 dpi. È preferibile usare i dp ai pixel in quanto si rende il layout più elastico per meglio adattarsi a schermi con diverse densità di pixel.

In questa immagine possiamo osservare la differenza tra i due casi, nel caso a sinistra viene utilizzato il valore wrap_content, mentre a destra match_parent.

Nel caso di sinistra possiamo apprezzare come il controllo si estenda in larghezza fino alla dimensione del testo contenuto, mentre a destra si espande fino a occupare lo spazio presente nel contenitore del pulsante stesso, cioè l'intera finestra.

Altre proprietà riguardano il testo, il colore del testo, i bordi del controllo, la posizione del controllo rispetto ad altri controlli ecc.

Alcune proprietà dei controlli possono essere modificate durante l'esecuzione richiamando specifici metodi.

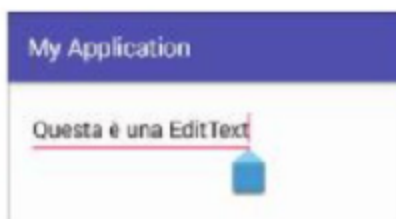
Widget di base

Vediamo ora alcuni controlli che possiamo definire di base: si tratta dei widget largamente utilizzati all'interno delle applicazioni.

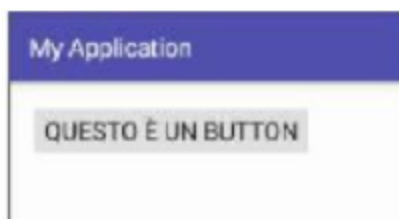
TextView

Gli oggetti TextView rappresentano le etichette di testo e appartengono alla classe android.widget.TextView. Il testo visualizzato può essere definito mediante il metodo setText(), che riceve come parametro una stringa.

EditText

Gli oggetti EditText rappresentano le caselle di testo e appartengono alla classe android.widget.EditText. Questo controllo estende la classe TextView e consente all'utente di immettere del testo. Il testo visualizzato può essere impostato mediante il metodo setText(), che riceve come parametro una stringa. Il metodo getText() invece, restituisce un oggetto di tipo android.text.Editable.

Gli oggetti Editable sono simili alle stringhe, e infatti implementano l'interfaccia java.lang.Char.Sequence.

Button

Rappresentano i pulsanti touch presenti sul display e appartengono alla classe android.widget.Button. Il controllo espande la classe TextView, per questo motivo possiamo impostare il testo mostrato al suo interno con il metodo setText() che riceve come parametro una stringa.

ESEMPI

Convertitore Dollaro/Euro • Vogliamo realizzare un convertitore da Euro a Dollari e viceversa utilizzando i tre widget seguenti:

- un pulsante (Button);
- una casella di testo (EditText);
- una etichetta di testo (TextView).

1. Per prima cosa passiamo a creare un nuovo progetto utilizzando una Empty Activity come activity principale.
2. Il layout è attualmente vuoto, per aggiungere i controlli necessari alla creazione della nostra applicazione dobbiamo aprire il file actvity_main.xml per aggiungere tutti i widget necessari per leggere i dati che l'utente ha digitato e per comunicare il risultato della conversione, come mostrato nell'immagine a lato: ▶
   Come possiamo notare osservando il codice XML (a pagina seguente), il nostro layout è composto da 2 etichette (TextView), 2 caselle di testo (EditText) e 2 pulsanti (Button). Dal codice XML possiamo anche ricavare gli id relativi a ogni controllo, che ci servono per poter leggere o scrivere informazioni relative a un determinato widget da codice.
   Mandando in esecuzione l'applicazione vediamo che è possibile scrivere all'interno delle caselle di testo EditBox; ovviamente facendo clic su uno dei due pulsante non succede nulla. Vedremo adesso come definire l'azione che deve essere eseguita quando il pulsante viene premuto.

activity_main.xml ×

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.example.emanuele.myapplication.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="USD"
        android:id="@+id/textView"
        android:textSize="25dp"
        ... />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:text="0"
        android:textSize="25dp"
        ... />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="EURO"
        android:id="@+id/textView2"
        android:textSize="25dp"
        ... />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:text="0"
        android:textSize="25dp"
        ... />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="USD -> EURO"
        android:id="@+id/button"
        android:textSize="20dp"
        android:onClick="usdToEuro"
        ... />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="EURO -> USD"
        android:id="@+id/button2"
        android:textSize="20dp"
        android:onClick="euroToUsd"
        ... />

</RelativeLayout>
```

3. Utilizzando il Project Explorer apriamo il file MainActivity.java e creiamo quindi due nuovi metodi all'interno del file: usdToEuro e euroToUsd. Analizziamo ora il codice del metodo usdToEuro: viene dapprima richiamata la funzione findViewById(String), la quale restituisce come oggetto la View avente l'id che è stato passato come parametro.

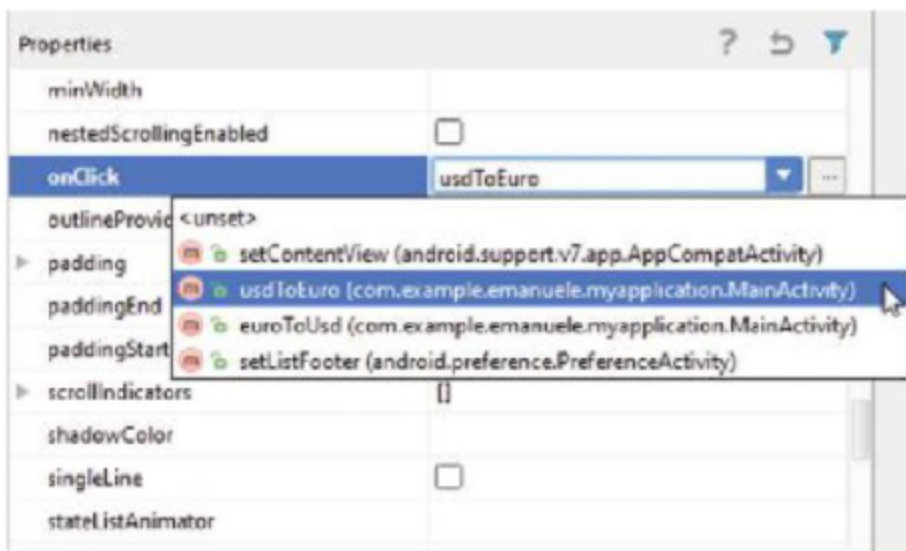
DEFINIZIONE

È la superclasse di tutti i widget, la funzione findViewById restituisce un oggetto di tipo View, bisogna poi fare un casting al tipo di widget che realmente è associato all'id passato come parametro.

In questo caso passiamo come parametro R.id.editText, poiché è l'id associato alla EditText che contiene il valore in dollari inserito dall'utente.
Una volta recuperato il riferimento alle due EditText leggiamo il valore in formato stringa inserito dall'utente utilizzando il metodo getText() e lo convertiamo nel tipo double utilizzando il metodo Double.parseDouble() della classe Double.
Calcoliamo infine il corrispettivo valore in euro e lo inseriamo nella EditText associata al valore in euro tramite il metodo setText().

MainActivity.java

```java
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    final double cambioEuroDollaro=1.1177;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void usdToEuro(View v){
        EditText edit_usd = (EditText)findViewById(R.id.editText);
        EditText edit_euro = (EditText)findViewById(R.id.editText2);
        double usd = Double.parseDouble(edit_usd.getText().toString());
        double euro = usd/cambioEuroDollaro;
        edit_euro.setText(String.valueOf(euro));
    }

    public void euroToUsd(View v){
        EditText edit_usd = (EditText)findViewById(R.id.editText);
        EditText edit_euro = (EditText)findViewById(R.id.editText2);
        double euro = Double.parseDouble(edit_euro.getText().toString());
        double usd = euro*cambioEuroDollaro;
        edit_usd.setText(String.valueOf(usd));
    }
}
```

4. Creare questi due metodi non basta affinché la nostra applicazione funzioni, bisogna infatti associare i suddetti eventi agli eventi onClick dei rispettivi pulsanti. Per fare ciò ci spostiamo nella visualizzazione grafica delle proprietà del controllo, selezioniamo la proprietà onClick e impostiamo il metodo creato in precedenza.

ZOOM

Utilizzo dei listener • È possibile realizzare lo stesso funzionamento istanziando un listener sull'evento onClick nell'onCreate dell'activity e definire allo stesso tempo i metodi onClick dei relativi pulsanti.

MainActivity.java ×

```
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    final double cambioEuroDollaro=1.1177;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button_usdToEuro = (Button) findViewById(R.id.button);
        button_usdToEuro.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                usdToEuro(v);
            }
        });
```

```
        Button button_euroToUsd = (Button) findViewById(R.id.button2);
        button_euroToUsd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                euroToUsd(v);
            }
        });
    }

    public void usdToEuro(View v){
        EditText edit_usd = (EditText) findViewById(R.id.editText);
        EditText edit_euro = (EditText) findViewById(R.id.editText2);
        double usd = Double.parseDouble(edit_usd.getText().toString());
        double euro = usd / cambioEuroDollaro;
        edit_euro.setText(String.valueOf(euro));
    }

    public void euroToUsd(View v){
        EditText edit_usd = (EditText)findViewById(R.id.editText);
        EditText edit_euro = (EditText)findViewById(R.id.editText2);
        double euro = Double.parseDouble(edit_euro.getText().toString());
        double usd = euro*cambioEuroDollaro;
        edit_usd.setText(String.valueOf(usd));
    }
}
```

5. Possiamo ora provare la nostra applicazione e verificare che essa funzioni correttamente.
Come possiamo vedere una volta inserito il valore in euro, cliccando sul bottone EURO → USD otteniamo la conversione in dollari visualizzata nella relativa EditText.

Prova adesso!

- Utilizzare il componente EditText
- Utilizzare il componente Button
- Utilizzare il componente TextView

Apri l'esempio Convertitore

1. Modifica il codice dell'esempio per fare in modo che la conversione avvenga automaticamente durante la digitazione del valore.
2. Modifica l'esempio inserendo una terza EditText associata al valore in sterline.
3. Modifica il codice aggiungendo un bottone che azzera i valori all'interno delle TextView e EditText.

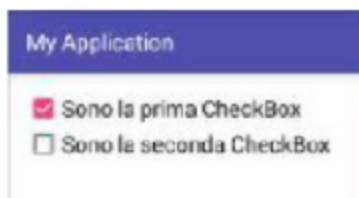
Altri widget molto utilizzati

ImageView

Rappresenta una immagine proveniente da un file che deve essere memorizzato in una cartella drawable. È presente in android.widget.ImageView. Il metodo principale che assegna un'immagine al controllo è setImageResource().

Dobbiamo inserire il solo nome del file dell'immagine e non l'estensione. Tuttavia l'estensione può essere solo .png oppure .jpg.

CheckBox

Rappresenta un controllo di tipo casella a scelta multipla ed è presente nella classe android.widget.CheckBox. Estende la classe Button e la classe TextView, infatti possiamo anche in questo controllo impostare il testo mostrato a fianco delle caselle di spunta, attraverso il metodo setText(). Il metodo isChecked() restituisce true o false a seconda che il controllo venga selezionato.

RadioButton

Rappresenta un controllo di tipo casella a scelta esclusiva ed è presente nella classe android.widget.RadioButton. Estende la classe Button e la classe TextView, infatti possiamo anche in questo controllo impostare il testo mostrato a fianco delle caselle di spunta, attraverso il metodo setText(). Il metodo isChecked() restituisce true o false a seconda che il controllo venga selezionato. Questi controlli possono essere raggruppati all'interno di un RadioGroup in modo che l'utente possa così attivare una sola delle opzioni del gruppo.

ToggleButton

My Application

ON OFF

Rappresentano degli interruttori toccabili sullo schermo e possiedono due stati: attivo (on) e non attivo (off). Sono presenti nella classe android.widget.ToggleButton. Lo stato è indicato dal metodo isChecked(), che restituisce true o false a seconda se il pulsante sia in stato on oppure off.

DatePicker

Rappresentano un controllo per agevolare l'utente nell'immissione di una data nel formato giorno, mese, anno. È un controllo presente nella classe android.widget.DatePicker. Possiamo recuperare la data impostata dall'utente grazie ai metodi getDayOfMonth(), getMonth() e getYear().

TimePicker

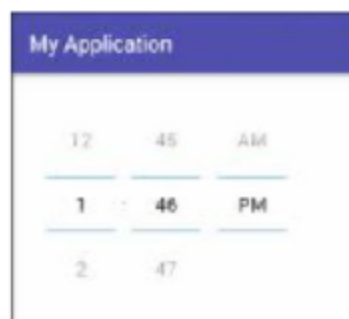
Rappresentano un controllo per agevolare l'utente nell'immissione di un orario nel formato ore, minuti. È un controllo presente nella classe android.widget.TimePicker. Possiamo recuperare l'orario impostato dall'utente grazie ai metodi getCurrentHour() e getCurrentMinute().

ListView

Rappresentano un controllo per agevolare la visualizzazione di numerosi elementi dello stesso tipo. È un controllo presente nella classe android.widget.ListView. Per riempirlo bisogna inizializzare un array di stringhe, creare un arrayadapter, e chiamare il metodo setAdapter() passando come parametro l'arrayadapter precedentemente creato.

ESEMPI

Generatore riassunto • Vogliamo realizzare un'applicazione che consenta all'utente di inserire dati quali nome, cognome, sesso, data di nascita ed eventuali campi d'interesse. L'applicazione deve inoltre poter generare un riassunto in formato testo delle informazioni inserite dall'utente. Vediamo i wigdet da utilizzare:

- EditText per consentire all'utente di inserire il proprio nome e cognome;
- RadioGroup contenente due RadioButton, che permettono all'utente di selezionare il proprio sesso;
- DatePicker, grazie al quale l'utente può facilmente selezionare la propria data di nascita;
- CheckBox, rappresentano possibili aree di interesse che l'utente può selezionare.

Modifichiamo il layout per ottenere l'aspetto desiderato.

Quando il bottone GENERA RIASSUNTO viene premuto bisogna leggere i valori dei vari widget e utilizzare i valori letti per generare un riepilogo delle informazioni in formato testo. A seconda del tipo di widget la lettura dei dati avviene con metodi diversi.

- EditText: il metodo getText() restituisce il testo all'interno della EditText.
- CheckBox: il metodo isChecked() restituisce true se la casella è spuntata, false altrimenti.
- RadioGroup: il metodo getCheckedRadioButtonId() restituisce l'id del RadioButton selezionato all'interno del gruppo. Grazie all'id possiamo recuperare un riferimento al RadioButton selezionato e ottenere informazioni a esso associate.

- DatePicker: il metodo getYear() restituisce l'anno selezionato, getMonth() restituisce il numero del mese selezionato (partendo da 0) e getDayOfMonth() restituisce il numero del giorno selezionato.

Combiniamo queste funzioni per ottenere un resoconto in formato testo delle informazioni inserite dall'utente. Visualizziamo il riassunto all'interno di una TextView.

```
MainActivity.java
package com.example.emanuele.myapplication;

import ...

public class MainActivity extends AppCompatActivity {
    String endl = "\r\n";

    EditText nome=null,cognome=null;
    RadioGroup sesso=null;
    DatePicker data=null;
    CheckBox tecnologia=null,arte=null,musica=null,sport=null,auto=null;
    TextView riassunto = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nome=(EditText) findViewById(R.id.editTextNome);
        cognome=(EditText) findViewById(R.id.editTextCognome);
        sesso=(RadioGroup)findViewById(R.id.radioGroup);
        data=(DatePicker)findViewById(R.id.datePicker);
```

```
        tecnologia=(CheckBox)findViewById(R.id.checkBoxTecnologia);
        arte=(CheckBox)findViewById(R.id.checkBoxArte);
        musica=(CheckBox)findViewById(R.id.checkBoxMusica);
        sport=(CheckBox)findViewById(R.id.checkBoxSport);
        auto=(CheckBox)findViewById(R.id.checkBoxAuto);

        riassunto=(TextView)findViewById(R.id.textViewRiassunto);
    }

    public void generaResoconto(View v){
        RadioButton selezionato = (RadioButton) findViewById(sesso.getCheckedRadioButtonId());
        String s="Nome: "+ nome.getText().toString() + endl
                + "Cognome: " + cognome.getText().toString()+ endl
                + "Sesso: " + selezionato.getText().toString() + endl
                + "Data di nascita: " + data.getDayOfMonth()
                + "/" + (data.getMonth()+1)
                + "/" + data.getYear() + endl
                + "Interessi: ";

        if(tecnologia.isChecked()) s+= "tecnologia ";
        if(arte.isChecked()) s+= "arte ";
        if(musica.isChecked()) s+= "musica ";
        if(sport.isChecked()) s+= "sport ";
        if(auto.isChecked()) s+= "auto ";

        riassunto.setText(s);
    }
}
```

Esempio di funzionamento: ▶

Prova adesso!

- Utilizzare il componente RadioButton
- Utilizzare il componente Button
- Utilizzare il componente RadioGroup

Apri l'esempio Generatore riassunto

1. Modifica il codice dell'esempio aggiungendo controlli per lo stato civile (celibe, sposato, divorziato, vedovo).
2. Modifica il codice aggiungendo controlli per le possibili lingue conosciute (italiano, inglese, francese, tedesco, spagnolo).
3. Modifica l'esempio affinché il colore del bottone cambi a seconda che sia stato selezionato Uomo o Donna.

Esercitiamoci

Test Vero/Falso

Indica, barrando la relativa casella, se le seguenti affermazioni sono vere o false.

1 I controlli TextView servono per ricevere le informazioni che l'utente digita tramite tastiera. V F
2 Per poter aggiungere un controllo di tipo EditText è necessario modificare il file del layout. V F
3 Non è possibile utilizzare il Drag & Drop per aggiungere controlli al layout. V F
4 I file relativi al layout si trovano nella cartella res. V F
5 È possibile modificare le proprietà dei controlli solo modificando a mano il file xml. V F
6 Ogni controllo è identificato da un id. V F
7 I dp sono un'unità di misura usata per indicare height e width dei controlli. V F
8 Wrap-content estende la larghezza del controllo fino a quella del contenitore padre. V F
9 Possiamo recuperare il testo scritto in una EditText utilizzando il metodo getContent. V F
10 I Button sono un'espansione della classe TextView. V F
11 Le ImageView possono essere utilizzate con qualsiasi tipo di immagine. V F
12 Posso assegnare lo stesso id a due controlli diversi. V F
13 Posso assegnare funzioni a eventi nel file del layout. V F
14 I listener attendono che un evento si verifichi. V F
15 La superclasse di tutti i widget si chiama Control. V F
16 Non è possibile utilizzare Button e RadioButton all'interno della stessa applicazione. V F
17 Il DatePicker viene utilizzato per agevolare l'utente nell'immissione di un orario. V F

Domande a risposta multipla

Indica la risposta corretta barrando la casella relativa.

1 Per recuperare un generico controllo da codice devo utilizzare il metodo
   a. getText
   b. findControl
   c. findView
   d. findViewById

2 Con quali tipi di immagine posso utilizzare la ImageView?
   a. gif
   b. jpg
   c. tif
   d. png

3 Quale metodo utilizzo per vedere se un RadioButton è stato selezionato?
   a. setText
   b. isChecked
   c. isEnabled
   d. isActive

4 Devo salvare le immagini utilizzate dalla ImageView nella cartella:
   a. layout
   b. scr
   c. drawable
   d. values

5 Il metodo getText della EditText restituisce un oggetto di tipo:
   a. String
   b. Char
   c. CharSequence
   d. Editable

Esercitiamoci

6 **Quale controllo implementa la funzionalità di interruttore ON/OFF?**

a. ToggleButton
b. Button
c. ChakcBox
d. RadioButton

7 **Per agevolare l'utente nell'immissione di un orario utilizziamo il controllo:**

a. TimePicker
b. EditText
c. ListView
d. DatePicker

8 **Quali controlli bisogna utilizzare per permettere all'utente di selezionare una sola delle opzioni?**

a. CheckBox
b. RadioButton
c. RadioGroup
d. TextView

9 **Per riempire una ListView utilizzo il metodo:**

a. setLayout
b. setText
c. setAdapter
d. getText

Problemi

Progetta e realizza completamente il codice in Java per Android che risolva il problema proposto.

1 Crea un progetto che legga da caselle di testo i dati necessari per calcolare l'area di un triangolo (base e altezza), quindi mostri il risultato in una terza casella di testo. Aggiungi anche un pulsante RadioButton per selezionare l'unità di misura del risultato (mm, cm, m, hm ecc.).

2 Crea un progetto che legga da caselle di testo i dati necessari per calcolare l'area di un trapezio (base minore, base maggiore e altezza), quindi mostri il risultato in una quarta casella di testo. Aggiungi anche un pulsante RadioButton per selezionare l'unità di misura del risultato (mm, cm, m, hm ecc.).

3 Crea un progetto che legga da caselle di testo i dati necessari per calcolare l'area di un cerchio (raggio o diametro), quindi mostri il risultato in una terza casella di testo. Aggiungi anche un pulsante RadioButton per selezionare l'unità di misura del risultato (mm, cm, m, hm ecc.).

4 Crea un progetto che dopo aver letto una data verifichi se essa appartiene a un anno bisestile o meno, stampando il risultato in un messaggio di testo di tipo toast.

5 Crea un progetto che dopo aver letto un numero in una casella di testo verifichi se il numero è pari o dispari, stampando il risultato in un messaggio di testo di tipo toast.

6 Crea un progetto che dopo aver letto una data di nascita calcoli e mostri in una casella di testo il giorno della settimana relativo, facendo attenzione a tener conto anche degli anni bisestili.

7 Crea un progetto che dopo aver letto due date, calcoli e stampi in una casella di testo quanti giorni sono intercorsi tra le due date, facendo attenzione a tener conto anche degli anni bisestili.

8 Crea un progetto che dopo aver letto un numero da una casella di testo verifichi se esso è un numero primo o meno visualizzando il risultato in una casella di testo.

9 Crea un progetto che dopo aver letto un numero da una casella di testo verifichi se esso è un numero perfetto o meno (un numero è perfetto quando è pari alla somma dei suoi divisori) visualizzando il risultato in una etichetta di testo.

10 Crea un progetto che mostri quattro pulsanti, indicanti le quattro operazioni, oltre a tre caselle di testo. L'utente scrive due numeri nelle caselle di testo, quindi facendo clic sul pulsante relativo all'operazione prescelta il programma calcola e mostra il risultato nella terza casella di testo.

11 Crea un progetto che mostri i primi 1000 numeri in una ListView, e che quando l'utente seleziona un numero venga visualizzato in una casella di testo se si tratta di un numero pari, dispari e primo.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Riconoscere i diversi widget utilizzabili nell'interfaccia grafica Android	○	○	○
Comprendere il ruolo del file manifest.xml	○	○	○
Comprendere il ruolo del file activity_main.xml	○	○	○
Capire il ruolo dei Listener	○	○	○
Comprendere il ruolo del metodo onClick	○	○	○
Comprendere il ruolo del metodo onCreate	○	○	○

Competenze	Scarso	Medio	Ottimo
Collocare i widget disponibili nel layout	○	○	○
Modificare le proprietà dei widget in ambiente grafico	○	○	○
Modificare le proprietà dei widget nel file activity_main.xml	○	○	○
Applicare il widget TextView	○	○	○
Applicare il widget EditText	○	○	○
Applicare il widget Button	○	○	○
Applicare il widget ImageView	○	○	○
Applicare il widget CheckBox	○	○	○
Applicare il widget RadioButton	○	○	○
Applicare il widget DatePicker	○	○	○
Applicare il widget TimePicker	○	○	○
Applicare il widget ListView	○	○	○
Utilizzare l'evento onCreate	○	○	○
Utilizzare l'evento onClick	○	○	○

5

LEZIONE

UN'APP COMPLETA: LA CALCOLATRICE

In questa lezione impareremo

- a passare parametri (sender) alle funzioni
- a modificare le proprietà dei controlli in run time

La calcolatrice

Utilizzando le conoscenze acquisite finora passiamo a realizzare una semplice calcolatrice, combinando i controlli EditText e Button per ottenere il layout indicato di seguito:

Per implementare il funzionamento della calcolatrice utilizziamo alcune variabili globali:

- input: è un oggetto di tipo EditText memorizza il riferimento alla casella di testo all'interno della quale l'utente inserisce i numeri;
- operator: di tipo char memorizza l'operazione che deve essere eseguita;
- temp: di tipo double memorizza il risultato parziale delle operazioni effettuate.

```
public class MainActivity extends AppCompatActivity {
    EditText input=null;
    char operator ='+';
    double temp=0;
```

Per dichiarare un oggetto di classe EditText utilizziamo la forma:

```
classe nome_oggetto=null;
```

dove la classe è in questo caso EditText, il nome dell'oggetto è input.

Suddividiamo i pulsanti in tre categorie, associando la stessa funzione all'evento onClick per tutti i pulsanti di quella categoria:

- numerici: fanno parte di questa categoria tutti i pulsanti che rappresentano le cifre da 0 a 9 e la virgola;
- operazionali: fanno parte di questa categoria i pulsanti che rappresentano le quattro operazioni fondamentali e il pulsante che rappresenta l'uguale;
- reset: fa parte di questa categoria solo il pulsante "C" (Clear).

Siccome più pulsanti richiamano la stessa funzione, utilizzeremo il parametro sender per riconoscere quale bottone ha invocato la funzione.
All'onClick dei pulsanti della categoria Numerici associamo la funzione append, che concatena la cifra rappresentata dal bottone al valore già presente nella casella di testo chiamata input.

DEFINIZIONE

Il sender è un parametro passato a una funzione che rappresenta l'oggetto che ha invocato la funzione stessa. Effettuando i dovuti casting è possibile ottenere tutte le informazioni associate al controllo chiamante.

```
public void append(View v){
    Button b = (Button)v;
    if (input.getText().toString().equals("0")){
        input.setText(b.getText());
    }else {
        input.getText().append(b.getText());
    }
}
```

La funzione non gestisce la possibilità di inserire due virgole di seguito poiché questo compito può essere delegato alla EditText andando a modificarne alcune proprietà.

Associamo ai pulsanti della categoria operazionali, che rappresentano le operazioni matematiche, la funzione operation, che provvede all'aggiornamento del risultato

```
    public void operation(View v){
        char nextOperator = ((Button)v).getText().charAt(0);
        Double numero = Double.parseDouble(input.getText().toString());

        switch (operator){
            case '+':
                temp+=numero;
                break;
            case '-':
                temp-=numero;
                break;
            case '*':
                temp*=numero;
                break;
            case '/':
                temp/=numero;
                break;
        }

        if (nextOperator=='='){
            input.setText(String.valueOf(temp));
        }else {
            input.setText("0");
        }

        operator=nextOperator;
    }
}
```

parziale e la variabile globale operator, che, come abbiamo detto prima, rappresenta la prossima operazione che deve essere eseguita. Nel caso in cui il pulsante che ha chiamato la funzione sia quello che rappresenta l'uguale allora viene collocato il risultato dell'operazione all'interno della casella di testo input, in caso contrario viene azzerata la casella input per permettere all'utente di inserire il secondo operando dell'operazione appena selezionata.

Infine dobbiamo associare il pulsante C alla funzione clear, che si occupa di azzerare il contenuto della EditText e di azzerare il risultato parziale memorizzato all'interno della variabile temp.

```
public void clear(View v){
    input.setText("0");
    operator='+';
    temp=0;
}
```

Un esempio di funzionamento è rappresentato nell'immagine a fianco: ▶

Prova adesso!

- Utilizzare il componente Button
- Utilizzare il componente EditText

Apri l'esempio Calcolatrice

1. Modifica il codice dell'esempio per aggiungere la funzione elevamento a potenza.
2. Modifica il codice aggiungendo la conversione dei valori da decimale a binario e viceversa.
3. Modifica l'esempio affinché ci sia la possibilità di memorizzare un risultato temporaneo per poterlo riutilizzare in futuro.
4. Crea una nuova versione della calcolatrice che consenta l'utilizzo delle parentesi.

Esercitiamoci

Problemi

Progetta e realizza completamente il codice in Java per Android che risolva il problema proposto.

1 Crea un progetto contenente tre pulsanti. Il clic su di essi deve aprire la stessa routine che mostra in un messaggio Toast contenente il testo che appare sul pulsante premuto.

2 Crea un progetto contenente una casella di testo e due pulsanti ai suoi lati con indicati i simboli "+" e "–". A ogni clic sul + il contenuto numerico della casella di testo deve essere incrementato di 1, altrimenti decrementato di 1.

3 Crea un progetto contenente due caselle di testo, un pulsante e due RadioButton. I due RadioButton consentono di scegliere se moltiplicare o dividere il contenuto delle due caselle di testo. Il risultato viene mostrato in un Toast al clic sul pulsante.

4 Crea un progetto contenente un pulsante e un oggetto DatePicker. Al clic sul pulsante deve essere mostrato il mese inserito nella TextView come testo del pulsante. Il secondo clic deve rimettere il testo originale sul pulsante.

5 Crea un progetto contenente due pulsanti. Al clic sul pulsante 1 si deve nascondere il pulsante 2, mentre al clic sul pulsante 2 si deve nascondere il pulsante 1. Al clic successivo sul pulsante si deve riattivare la visione dell'altro pulsante. Utilizzare un solo metodo onClick che deve riconoscere da quale pulsante è provenuto il clic.

6 Dato un elenco di automobili presenti in una serie di CheckBox (non più di 10), dopo il clic su di un pulsante l'applicazione deve scrivere i nomi di quelle selezionate in una stringa da porre in una casella di testo. Aggiungi anche un pulsante che azzeri le scelte effettuate.

7 Crea un progetto in grado di mostrare una immagine a seconda della parola selezionata da 3 RadioButton (Cane, Gatto, Criceto).

8 Crea un progetto che contenga 3 caselle di testo e un pulsante. La prima casella contiene un imponibile, la seconda l'imposta in percentuale da applicare e la terza deve dare il risultato, calcolando l'imponibile più l'imposta.

9 Crea un progetto contenente due caselle di testo, una TextView e quattro pulsanti. La didascalia dei quattro pulsanti è rispettivamente: "+" per il primo, "–", per il secondo, "*" per il terzo e "/" per il quarto. A ogni clic su di un pulsante il programma deve effettuare l'operazione matematica tra i valori presenti nelle due caselle di testo e fornire il risultato nella TextView.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Capire il ruolo dei sender	○	○	○
Comprendere il ruolo del metodo onClick	○	○	○
Comprendere il ruolo del metodo onCreate	○	○	○

Competenze	Scarso	Medio	Ottimo
Collocare i widget disponibili nel layout	○	○	○
Istanziare un oggetto in fase di run time	○	○	○
Applicare il parametro sender	○	○	○
Applicare il widget EditText	○	○	○
Applicare il widget Button	○	○	○
Utilizzare l'evento OnCreate	○	○	○
Utilizzare l'evento OnClick	○	○	○
Applicare il file manifest.xml	○	○	○

6

LEZIONE

I SENSORI

In questa lezione impareremo

- a utilizzare un servizio di sistema
- a recuperare un riferimento a un sensore
- a leggere dati da un sensore

SensorManager

I dispositivi Android integrano numerosi sensori che permettono al dispositivo di rilevare informazioni legate ai movimenti, alla posizione, alle condizioni ambientali.

DEFINIZIONE

Il sensore è un trasduttore che si trova in diretta interazione con il sistema misurato ed è riferito solamente al componente che fisicamente effettua la trasformazione della grandezza d'ingresso in un segnale di natura elettrica.

I sensori vengono utilizzati nel campo della programmazione in diversi ambiti: ad esempio possiamo utilizzare il sensore di temperatura per tener traccia della temperatura di casa, il giroscopio e l'accelerometro per rilevare i movimenti dell'utente, il sensore magnetico per riconoscere in quale direzione punta il dispositivo ecc.

I sensori possono essere classificati in tre categorie:

- sensori ambientali: rilevano informazioni riguardo l'ambiente circostante quali pressione, temperatura e umidità;
- sensori di posizione: rilevano la posizione geografica del dispositivo;
- sensori di movimento: rilevano le forze fisiche che agiscono sul dispositivo. Tra questi abbiamo accelerometro e giroscopio.

ZOOM

I sensori dello smartphone

Accelerometro

L'accelerometro è un componente, fatto di silicio e costituito da un minuscolo involucro con all'interno una massa. Quest'ultima ha una certa flessibilità che consente di valutarne gli spostamenti senza dover utilizzare una molla, come succede negli accelerometri di grandi dimensioni. La massa è costituita da piccole lamelle mobili che si muovono tra una serie di lamelle fisse.
Poiché le lamelle mobili non possono toccare le lamelle fisse, si è pensato di misurare la differenza di potenziale tra la lamella mobile e le due lamelle fisse; la lamella mobile e quella fissa fungono quindi da armature del condensatore. La differenza di potenziale che si ha tra la lamella mobile e la lamella fissa varia in funzione della distanza tra le due lamelle. L'accelerometro è il sensore più utile per lo smartphone; Apple fu la prima a scoprirne i vantaggi, usando questo sensore per orientare l'immagine dello schermo in base all'orientazione del dispositivo.

Giroscopio

Lo scopo principale del giroscopio è quello di mantenere l'orientamento grazie alla conservazione del momento angolare. Così come un oggetto tende a conservare il suo stato di moto rettilineo uniforme, un oggetto ruotante tenderà a mantenere il suo moto rotatorio nella stessa direzione. Grazie a questa proprietà, il giroscopio è in grado di rilevare la rotazione relativa dell'oggetto cui è solidale, descrivendone la

variazione nel corso del tempo. Il giroscopio di uno smartphone è in grado pertanto di rilevare la rotazione su tutti e tre gli assi dello spazio.

Magnetometro

Il magnetometro è lo strumento di misura del campo magnetico e simula, nello smartphone, una bussola. Le applicazioni più interessanti che usano il magnetometro sono quelle per navigazione satellitare in tempo reale.

Sensore di prossimità

I sensori di prossimità sono dei sensori in grado di rilevare la presenza di oggetti nelle immediate vicinanze del lato sensibile del sensore stesso, senza che vi sia un effettivo contatto fisico. La distanza entro cui questi sensori rilevano oggetti dipende esclusivamente dalla tipologia del sensore e dalla loro qualità progettuale. Negli smartphone vengono usati i sensori di prossimità ottici; la loro tecnologia si basa sulla rilevazione della riflessione di fasci di raggi infrarossi opportunamente scelti per non avere interferenze con la luce ambientale. Il fascio viene riflesso dalla superficie stessa dell'oggetto rilevato, per lo stesso fenomeno per cui la luce visibile può essere riflessa e percepita dai nostri occhi.
L'utilità di questo sensore sta nel fatto di disattivare lo schermo dello smartphone quando esso si avvicina all'orecchio durante una conversazione oppure quando viene messo in tasca; in questo modo non si verificheranno tocchi indesiderati sullo schermo.

Sensore di luminosità

Il sensore di luminosità ambientale ha il compito di preservare la durata della batteria variando il livello di retroilluminazione del display LCD in funzione della luce che viene a contatto con il dispositivo.

Sensore di temperatura e umidità (termometro/igrometro)

Il termometro negli smartphone è un sensore che rileva la temperatura e l'umidità.

Barometro

Il barometro è un sensore che misura la pressione atmosferica. Nello smartphone il barometro serve a migliorare il fix GPS.

Pedometro

Il pedometro non rientra nella categoria dei sensori ma è un apparato utilizzato per misurare il numero di passi e pertanto può fornire una misura indiretta della distanza, velocità e il passo di una persona. Al suo interno vi è un sensore capace di registrare il movimento corporeo fatto a ogni passo.

Sensore di impronte

Apple introduce per la prima volta in uno smartphone il primo sistema di riconoscimento biometrico, quale il sensore d'impronte, rivoluzionando quindi il modo di vedere la sicurezza negli smartphone.
Il sensore d'impronte permette, con il semplice contatto di un dito sul tasto centrale del dispositivo, l'autenticazione di una persona.

I sensori possono essere sia hardware che software, ma Android nasconde questa distinzione, permettendoci di utilizzarli trascurandone la loro natura.
Il numero, il tipo e la qualità dei sensori variano da dispositivo a dispositivo, ma alcuni sensori quali accelerometro, giroscopio e sensore di prossimità sono sempre presenti anche nei dispositivi più economici.
Vediamo come leggere i valori rilevati dai sensori che il nostro dispositivo integra.
Android permette di collegarci ai sensori solo utilizzando un servizio di sistema, chiamato SENSOR_SERVICE, che ci fornisce un oggetto di tipo SensorManager.

```
SensorManager manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Una volta ottenuto il riferimento possiamo chiedere al SensorManager la lista di tutti i sensori di cui il dispositivo è dotato.

```
List<Sensor> listaSensori = manager.getSensorList(Sensor.TYPE_ALL);
```

Otteniamo una lista di oggetti di tipo Sensor.

Vediamo l'elenco dei vari sensori, ciascuno definito come costante della classe Sensor:

```
TYPE_PROXIMITY:                //Sensore di prossimità
TYPE_AMBIENT_TEMPERATURE:      //Sensore di temperatura
TYPE_GYROSCOPE:                //Giroscopio
TYPE_LIGHT:                    //Sensore di luminosità
TYPE_ACCELEROMETER:            //Accelerometro
TYPE_LINEAR_ACCELERATOR:       //Sensore di accelerazione lineare
TYPE_MAGNETIC_FIELD:           //Sensore di campo magnetico
TYPE_PRESSURE:                 //Sensore barometrico
```

La classe Sensor

La classe Sensor ci permette di ottenere svariate informazioni riguardo al sensore che essa rappresenta, in particolare integra dei metodi molto interessanti:

- getName restituisce il nome completo del sensore;
- getPower restituisce il consumo in mAh del sensore quando in uso;

- getType restituisce il tipo generico del sensore;
- getVendor restituisce il nome della ditta che produce il sensore;
- getVersion restituisce la versione del sensore;
- getResolution restituisce la risoluzione del sensore espressa nella relativa unità di misura.

Possiamo chiedere al manager il sensore di default di un determinato tipo:

```
Sensor s = manager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

Per poter leggere i dati da un sensore bisogna utilizzare un ascoltatore (Listener), ovvero un oggetto che continuamente ascolta ciò che il sensore comunica al dispositivo e ne elabora le informazioni per renderle disponibili lato codice.
Il Listener *deve* implementare i metodi onAccuracyChanged e onSensorChanged della classe SensorEventListener. All'interno del metodo onSensorChanged possiamo utilizzare il parametro event di tipo SensorEvent per recuperare i valori letti dal sensore e utilizzarli a nostro piacere.

I valori rilevati del sensore sono memorizzati all'interno di un array di 3 elementi di tipo float. L'array in questione è memorizzato all'interno dell'oggetto di tipo SensorEvent.
Da notare che non tutti gli elementi vengono sempre utilizzati, ma il numero di elementi utilizzati varia a seconda del tipo di sensore.
Nell'accelerometro per esempio vengono usati tutti gli elementi, poiché l'accelerometro deve funzionare lungo i tre assi dello spazio, ma se si pensa al sensore di prossimità, che deve solo rilevare la distanza lungo un unico asse, questo utilizzerà solo uno dei tre elementi mentre i restanti 2 rimarranno a 0.

```
private class ListenerSensore implements SensorEventListener{

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }

    @Override
    public void onSensorChanged(SensorEvent event) {

    }
}
```

Per iniziare a leggere dati da un determinato sensore bisogna istanziare un oggetto della classe ListenerSensore e utilizzare il metodo registerListener della classe SensorManager per collegare l'istanza del listener a un determinato sensore.

```
ListenerSensore listener = new ListenerSensore();
manager.registerListener(listener, s, SensorManager.SENSOR_DELAY_NORMAL);
```

Per terminare la lettura dei dati di un determinato sensore è sufficiente eliminare la registrazione del Listener a esso associato utilizzando il metodo unregisterListener

```
manager.unregisterListener(listener,s);
```

Vediamo un semplice esempio dove leggiamo i valori dell'accelerometro e li mostriamo continuamente all'utente.

ESEMPIO

Accelerometro • Creiamo un nuovo progetto e inseriamo quattro controlli di classe TextView nel layout; attraverso questi componenti visualizziamo alcune informazioni riguardo al sensore selezionato. Mostriamo, in particolare, il nome completo e i valori letti dal sensore preso in esame.

Per semplificare il codice è necessario che l'activity stessa implementi la classe SensorEventListener.
Dichiariamo le variabili necessarie e le inizializziamo nell'onCreate.

```
MainActivity.java ×
package com.example.emanuele.sensore;

import ...

public class MainActivity extends AppCompatActivity implements SensorEventListener {
    TextView nome, value0, value1, value2;
    SensorManager manager;
    Sensor accelerometro;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nome = (TextView) findViewById(R.id.nome);
        value0 = (TextView) findViewById(R.id.value0);
        value1 = (TextView) findViewById(R.id.value1);
        value2 = (TextView) findViewById(R.id.value2);
```

Utilizzando il metodo getSystemService recuperiamo l'oggetto di tipo SensorManager, e con il metodo getDefaultSensor otteniamo il riferimento al sensore di tipo TYPE_ACCELEROMETER.

```
        manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometro = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }
```

Definiamo i metodi necessari per implementare la classe SensorEventListener:
- onAccuracyChanged: questo metodo rimane vuoto in quando non è nostro interesse effettuare azioni in conseguenza a un cambio di accuratezza delle misure;
- onSensorChanged: all'interno del metodo recuperiamo i nuovi valori letti dal sensore e li inseriamo nelle TextView per mostrarli all'utente.

```
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        value0.setText("X: " + event.values[0]);
        value1.setText("Y: " + event.values[1]);
        value2.setText("Z: " + event.values[2]);
    }
```

La lettura dei dati da un sensore è una procedura abbastanza onerosa in termini di risorse, quindi è buona norma *spegnere* il Listener quando l'applicazione non è più visibile, e *riaccenderlo* una volta che l'applicazione torna attiva.
Per avviare il Listener bisogna registrarlo utilizzando il metodo registerListener della classe SensorManger. Implementiamo questa procedura nel metodo onResume dell'activity.

```
    @Override
    protected void onResume() {
        super.onResume();

        if (accelerometro != null) {
            manager.registerListener(this, accelerometro, SensorManager.SENSOR_DELAY_NORMAL);
            nome.setText(accelerometro.getName());
        } else {
            nome.setText("Sensore del tipo richiesto non presente!");
        }
    }
```

All'interno del metodo onPause dell'activity utilizziamo il metodo unregisterListener per *disattivare* il Listener sul sensore selezionato.

```java
@Override
protected void onPause() {
    super.onPause();

    if (accelerometro != null) {
        manager.unregisterListener(this, accelerometro);
    }
}
```

Vediamo un esempio di funzionamento dell'applicazione appena realizzata: ▶

Sensore

K3DH Acceleration Sensor
X: -0.7354988
Y: 1.9885708
Z: 9.765789

Cambiando il tipo di sensore richiesto possiamo visualizzare informazioni relative ad altri sensori: vediamo ad esempio il sensore di prossimità, che fa uso di solo uno dei tre valori: ▶

Alcuni sensori possono non essere presenti nel dispositivo. Richiedendo infatti il sensore di temperatura ambientale, di cui il dispositivo utilizzato è sprovvisto otteniamo il messaggio di errore impostato: ▶

Prova adesso!

- Utilizzare la classe SensorManager
- Utilizzare la classe Sensor

Apri l'esempio Accelerometro

1. Modifica il codice per far si che l'applicazione avvisi l'utente quando il telefono è in posizione verticale.
2. Modifica il codice in modo che riconosca quando il dispositivo è rimasto fermo per un certo periodo.

Esercitiamoci

Test Vero/Falso

Indica, barrando la relativa casella, se le seguenti affermazioni sono vere o false.

1. I sensori possono essere solo di tipo hardware. V F
2. Per collegarci ai sensori dobbiamo utilizzare un servizio di sistema. V F
3. Il SensorManager permette di recuperare la lista di tutti i sensori. V F
4. La lista dei sensori è uguale per ogni dispositivo. V F
5. La classe Sensor permette di ottenere informazioni riguardo a un sensore. V F
6. Android non fa distinzioni tra i tipi di sensori. V F
7. Possiamo leggere valori da sensore usando la classe Sensor. V F
8. Bisogna utilizzare dei Listener per leggere dati dal sensore. V F
9. La classe SensorManager si occupa della registrazione dei Listener. V F
10. Non posso impostare la velocità con la quale i dati vengono letti. V F
11. I Listener utilizzano poche risorse di sistema. V F
12. I Listener vanno spenti quando l'applicazione non è più visibile V F

Domande a risposta multipla

Indica la risposta corretta barrando la casella relativa.

1 L'accelerometro appartiene alla categoria dei sensori:

a. ambientali
b. di movimento
c. di posizione
d. nessuna delle precedenti

2 Quanti elementi ha l'array nel quale vengono memorizzate le misure fornite da un sensore?

a. 1 b. 2 c. 3 d. 4

3 Le misure fornite dal sensore sono memorizzate in un array di tipo:

a. int c. char
b. float d. double

4 Quali di questi metodi fanno parte delle classe SensorEventLinstener?

a. onCreate
b. onAccurracyChanged
c. onSensorChanged
d. nessuno dei precedenti

Associazione

Associa ciascun metodo (a sinistra) con i dati che esso restituisce (a destra), in relazione alla classe Sensor.

A. getName	.... restituisce il tipo generico del sensore
B. getPower	.... restituisce la versione del sensore
C. getType	.... restituisce il nome della ditta che produce il sensore
D. getVendor	.... restituisce il nome completo del sensore
E. getVersion	.... restituisce la risoluzione del sensore
F. getResolution	.... restituisce il consumo in mAh del sensore quando in uso

Esercitiamoci

Problemi

Progetta e realizza completamente il codice in Java per Android che risolva il problema proposto.

1. Crea un progetto contenente tre RadioButton, ognuno rappresentante un sensore diverso, e tre TextView, una per ogni elemento dell'array di lettura dati. L'utente deve visualizzare nelle TextView i valori relativi al sensore selezionato.

2. Crea un progetto che legga i dati dall'accelerometro e li analizzi in modo da riconoscere quando l'utente scuote il dispositivo. In tal caso visualizza un Toast contente un messaggio di avvenuto riconoscimento.

3. Crea un progetto contente una ImageView. Il controllo deve contenere un'immagine di una lampadina accesa quando il dato fornito dal sensore di luminosità scende sotto una determinata soglia, altrimenti mostra l'immagine di una lampadina spenta.

4. Crea un progetto che legga i dati del sensore contapassi e visualizzi all'interno di una TextView i passi fatti dall'avvio dell'applicazione. Deve inoltre essere possibile azzerare il conteggio.

5. Crea un progetto che legga i dati del giroscopio/accelerometro e che comunichi all'utente tramite una TextView quando il telefono è perfettamente in piano. Deve inoltre essere possibile modificare il livello di sensibilità.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Riconoscere i diversi sensori esistenti in un dispositivo mobile	○	○	○
Individuare le grandezze lette dai vari sensori	○	○	○
Conoscere il servizio SENSOR_SERVICE	○	○	○
Comprendere il ruolo della classe Sensor	○	○	○
Comprendere il ruolo della classe SensorManager	○	○	○

Competenze	Scarso	Medio	Ottimo
Saper rilevare valori letti dai sensori	○	○	○
Utilizzare la classe Sensor	○	○	○
Utilizzare la classe SensorManager	○	○	○
Associare un Listener ai sensori	○	○	○
Creare applicazioni che utilizzino l'accelerometro	○	○	○
Creare applicazioni che utilizzino il sensore di prossimità	○	○	○
Creare applicazioni che utilizzino il sensore di temperatura	○	○	○

7

LEZIONE

DATABASE LOCALI

In questa lezione impareremo

- a creare un nuovo database locale
- a creare una tabella
- a inserire, modificare ed eliminare record

Il salvataggio dei dati

Un sistema operativo per sistemi fissi, come ad esempio per i personal computer, fornisce un file system che consente alle applicazioni di memorizzare e rileggere file. Rispetto a un file system per computer, che consente di assegnare permessi di accesso alla singola risorsa, in Android esistono solo due tipologie di accesso: i dati possono essere di tipo privato, quindi gestibili solo dall'applicazione stessa, oppure di tipo pubblico, quindi accessibili da tutte le altre applicazioni.

Vediamo i principali tipi di memorizzazione permanente offerti da Android:

- Shared Preferences (dati pubblici/privati);
- Internal Storage (dati privati);
- External Storage (dati pubblici);
- SQLite Database (dati strutturati privati);
- Rete (web service);
- Content Provider (dati privati accessibili).

Una applicazione può scrivere e leggere valori, chiamati Preferences, condivisi con altre applicazioni dello stesso package (shared preferences) oppure privati dell'activity. Le Preferences sono le informazioni che consentono di personalizzare il sistema, come ad esempio la suoneria predefinita, il volume, il font, lo sfondo ecc.

Le Preferences possono essere definite da una applicazione mediante il codice oppure modificando uno specifico file XML da salvare nella cartella res\xml.

Nei prossimi paragrafi vedremo come gestire i dati permanenti delle applicazioni attraverso il database SQLite.

SQLite

SQLite è un dbms relazionale open source supportato da Android: per utilizzarlo non è necessaria alcuna installazione o configurazione. Supporta tipi di dato text, integer e real, inoltre ogni database SQLite è privato: se una app vuole gestirne il contenuto lo potrà fare attraverso i Content Provider.

Tutti i database presenti sul dispositivo mobile sono memorizzati nella directory:
/data/data/<NOME_PACKAGE>/databases

Vediamo in questo caso dove è collocato, nella struttura presente sul dispositivo, un database di nome myfriendsDB:

Il codice SQL utilizzabile in questo database è compatibile in gran parte con lo standard SQL-92.

DEFINIZIONE

Lo standard per SQL chiamato SQL-92, implementato nel 1992, dagli istituti ANSI e ISO, indica una versione abbastanza elementare di SQL: non utilizza gli oggetti e i trigger, inoltre non è computazionalmente completo, in quanto mancano le istruzioni principali di controllo.

La classe SQLiteOpenHelper

Vediamo adesso le classi principali messe a disposizione da Android per la gestione dei database SQLite. La classe SQLiteOpenHelper implementa i principali metodi necessari alla creazione e l'aggiornamento del database, vediamone la sintassi principale, in questo caso creiamo la nostra classe di esempio (chiamata DbHelper) che eredita dalla classe SQLiteOpenHelper:

```
//Classe di esempio (DbHelper)
public class DbHelper extends SQLiteOpenHelper
{
   //Proprietà nome e versione del database
   final static String DB_NAME = "nome_db";
   final static int DB_VERSION = 1;
   //Riscrittura costruttore
   public DbHelper(Context context)
   {
       ...
   }
   //Riscrittura metodi Oncreate e OnUpgrade
   ...
} //Fine classe DbHelper
```

Possiamo notare che esistono due proprietà (DB_NAME) e (DB_VERSION) di tipo final che contengono rispettivamente, la prima il nome del database e la seconda la versione; questi vanno passati mediante il metodo super, riscrivendo il metodo costruttore, nel modo seguente:

```
public DbHelper(Context context)
{
   super(context, DB_NAME, null, DB_VERSION);
}
```

Vediamo i metodi principali di questa classe, che vanno riscritti in overriding:

- onCreate(SQLiteDatabase db): questo metodo viene eseguito quando il database viene creato per la prima volta. Possiamo scrivere al suo interno le query SQL necessarie alla creazione delle tabelle del database, vediamo il codice di esempio che ne illustra la sintassi:

```
public void onCreate(SQLiteDatabase db)
{
   String sql = "CREATE TABLE promemoria(";
   sql = sql + "ID INTEGER PRIMARY KEY,";
   sql = sql + "nome VARCHAR NOT NULL,";
```

```
    sql = sql + "giorno_ora VARCHAR NOT NULL,";
    sql = sql + "testo VARCHAR NOT NULL,";
    sql = sql + ")";
    //Metodo execSQL() esegue la query
    db.execSQL(sql);
}
```

In questo caso possiamo notare che viene creata una tabella di nome promemoria con quattro campi (ID, nome, giorno_ora, testo).

- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion): questo metodo viene eseguito quando il database viene aggiornato. I parametri oldVersion e newVersion vengono utilizzati per il controllo di versione, vediamo il codice di esempio che ne illustra la sintassi:

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion)
{
    //Aggiornamento tabelle
}
```

- onOpen(SQLiteDatabase db): metodo che viene eseguito appena il database viene aperto. Utile inserire al suo interno eventuali query di controllo sui dati già presenti nel database.

Creiamo ora la classe DbHelper che estende SQLiteOpenHelper ed effettua l'override dei metodi necessari al funzionamento. Vediamo questo esempio di codice che mostra la creazione di una tabella di un promemoria, all'interno del metodo onCreate, che verrà eseguito alla creazione del database. Possiamo notare come, in questo caso i nomi dei campi vengano sostituiti dalle costanti definite dalla classe DatabaseStrings:

```
public class DbHelper extends SQLiteOpenHelper {
    public static final String DBNAME="DATABASE1";
    public DbHelper(Context context) {
        super(context, DBNAME, null, 1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String q="CREATE TABLE "+
                DatabaseStrings.TBL_NAME + " ( "+
                DatabaseStrings.FIELD_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
                DatabaseStrings.FIELD_TITOLO + " VARCHAR," +
                DatabaseStrings.FIELD_DESCRIZIONE + " VARCHAR)";
        db.execSQL(q);
    }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Come avrete notato viene utilizzata l'annotazione `@Override` prima del metodo onCreate e del metodo onUpgrade. Si tratta di una annotazione aggiunta che viene usata in Java per indicare al compilatore che il metodo su cui è posta vuole effettuare un override (sovrascrittura) di un metodo della sua superclasse.

Nella classe DatabaseStrings abbiamo salvato alcune costanti (possiedono l'attributo final) utili nell'uso del database. Assegniamo come nome del campo (FIELD_ID) la stringa "ID", come titolo del campo (FIELD_TITOLO) la stringa "titolo" ecc. È bene tenere i nomi delle tabelle e i nomi dei loro attributi salvati in un'altra classe. Questi accorgimenti permettono di ridurre il numero di eccezioni SQL dovute a una errata digitazione dei nomi di tabelle o attributi.

```
public class DatabaseStrings {
    public static final String FIELD_ID = "ID";
    public static final String FIELD_TITOLO = "titolo";
    public static final String FIELD_DESCRIZIONE = "descrizione";
    public static final String TBL_NAME = "promemoria";
}
```

Il riferimento alla classe di esempio creata sopra (DbHelper) ci consentirà di operare direttamente sul database, tale riferimento può essere recuperato utilizzando due metodi differenti:

- getReadableDatabase restituisce un oggetto di tipo SQLiteDatabase che rappresenta il database in formato read only;
- getWritableDatabase restituisce un oggetto di tipo di SQLiteDatabase grazie al quale possiamo inserire nuovi record oppure leggere e/o modificare quelli esistenti.

Quindi all'interno dell'activity della nostra applicazione dovremo utilizzare i metodi visti sopra nelle seguenti modalità, per aprire il database in sola lettura e in lettura scrittura:

```
//Apertura database in sola lettura
SQLiteDatabase db = db.getReadableDatabase();
...
//Apertura database in lettura e scrittura
SQLiteDatabase db = db.getWriteableDatabase();
```

Alla prima chiamata in assoluto del metodo getWritableDatabase viene eseguito il metodo onCreate.

La gestione del database

La classe SQLiteDatabase gestisce le informazioni contenute nel database, vediamone i metodi principali:

- execSQL (String sql, Object[] bindArgs)

  esegue la query SQL contenuta nella stringa passata come parametro, oltre a eventuali parametri; non restituisce alcun valore;
- insert (String table, String nullColumnHack, ContentValues values)

  permette di inserire nuove tuple all'interno di una determinata tabella. I valori che ogni attributo deve assumere sono specificati utilizzando il parametro di tipo ContentValues;
- delete (String table, String whereClause, String[] whereArgs)

  elimina i record specificati. I record da eliminare vengono selezionati utilizzando la where clause ed eventuali argomenti. Restituisce un intero che indica il numero di tuple eliminate;
- update (String table, ContentValues values, String whereClause, String[] whereArgs)

  aggiorna i record selezionati. I nuovi valori per le tuple selezionate sono specificati utilizzando il parametro values. Il metodo restituisce un intero che rappresenta il numero di tuple che sono state aggiornate;
- query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

  restituisce un oggetto Cursor contenente i risultati;
- rawQuery (String sql, String[] selectionArgs)

  richiede in ingresso una query SQL e ritorna un Cursor contenente eventuali dati di ritorno;

Cursor e ContentValues

Cursor è la classe che fornisce l'accesso al result set di una query. Può essere vista come una lista concatenata di record, ciascuno dei quali rappresenta una distinta tupla risultato.

DEFINIZIONE

Il termine è piuttosto generico, in campo informatico può anche indicare un insieme di valori restituiti da un metodo o da una funzione. Nel caso specifico dei database il result set rappresenta l'insieme di record o di tuple ottenute eseguendo una query SQL.

La classe Cursor inoltre fornisce i metodi necessari al recupero delle informazioni sugli attributi e sui valori a essi associati.

- getColumnCount()
  restituisce il numero di attributi che compongono il result set
- getColumnName(int columnIndex)
  restituisce il nome dell'attributo corrispondente all'indice specificato
- getColumnNames()
  restituisce un array di stringhe contente i nomi degli attributi
- getColumnIndex(String columnName)
  restituisce l'indice che identifica l'attributo con il nome specificato
- getCount()
  restituisce il numero totale di tuple che compongono il result set
- moveToPosition(int position)
  sposta il cursore sul record alla posizione specificata
- moveToNext()
  sposta il cursore al record successivo
- getString(int columnIndex)
  restituisce il valore dell'attributo all'index specificato per il record corrente

La classe ContentValues rappresenta una mappa di valori ed è principalmente utilizzata per l'inserimento di una nuova tupla in una tabella. Infatti ogni istanza della classe può contenere numerose coppie chiave valore, dove le chiavi rappresentano i nomi degli attributi mentre il valore rappresenta il dato a essi assegnato. La classe ContentValues mette a disposizione i seguenti metodi:

- put (String key, String value)
  viene utilizzato per inserire una nuova coppia chiave/valore o, nel caso la chiave identifichi una coppia già esistente, per modificarne il valore;
- getAsString (String key)
  restituisce il valore di tipo stringa che si trova in corrispondenza della chiave specificata. Esistono numerosi metodi per inserire e recuperare valori di diverso tipo;

Implementiamo quindi una nuova classe DbManager che incapsula DbHelper andando a dichiarare le intestazioni dei metodi che ci permettono di aggiungere e selezionare informazioni dal database.

DbManager.java ×

```
package com.example.emanuele.promemoria;

import ...

public class DbManager {
    private DbHelper dbhelper = null;

    public DbManager(Context context){
        dbhelper= new DbHelper(context);
    }

    public boolean delete(int id)    {...}

    public boolean save(String titolo, String descrizione){...}

    public Cursor getAllRecords(){...}

    public class DbHelper extends SQLiteOpenHelper {...}

}
```

Abbiamo visto che con la nostra classe di esempio DbHelper possiamo effettuare qualsiasi modifica al database. Per evitare abusi occorre quindi dichiarare la classe DbHelper all'interno di un'altra classe, in questo caso DbManager, in modo da incapsularla e nasconderne il funzionamento. In tal modo sarà dunque impossibile recuperare dei riferimenti al database e operare direttamente su esso.

Dobbiamo utilizzare i metodi della classe DbHelper per aggiungere, eliminare o recuperare dei promemoria dal database.

- Aggiungere un promemoria: dichiariamo il metodo save che vuole come parametri il titolo e la descrizione del promemoria da aggiungere. Il metodo recupera un riferimento al database tramite il metodo getWritableDatabase, crea un nuovo oggetto ContentValues e chiama la funzione insert dell'oggetto SQLiteDatabase. La funzione restituisce un booleano che indica se l'inserimento è avvenuto.

Da notare che utilizzando la funzione appena dichiarata andiamo a nascondere come il nuovo promemoria viene realmente inserito all'interno del database.

```
public boolean save(String titolo, String descrizione){
    SQLiteDatabase db=dbhelper.getWritableDatabase();

    ContentValues cv=new ContentValues();
    cv.put(DatabaseStrings.FIELD_TITOLO, titolo);
    cv.put(DatabaseStrings.FIELD_DESCRIZIONE, descrizione);

    if (db.insert(DatabaseStrings.TBL_NAME, null,cv)!=-1)
        return true;
    else
        return false;
}
```

- **Eliminare** un promemoria: realizziamo la funzione delete che riceve in ingresso l'id del promemoria da rimuovere e restituisce un booleano che indica se l'eliminazione è avvenuta con successo.

```
public boolean delete(int id)    {
    SQLiteDatabase db=dbhelper.getWritableDatabase();

    if (db.delete(DatabaseStrings.TBL_NAME, DatabaseStrings.FIELD_ID+"=?",
            new String[]{String.valueOf(id)})>0)
        return true;
    else
        return false;
    }
```

- **Recuperare** tutti i promemoria: la funzione getAllRecords restituisce il Cursor che contiene tutti i promemoria presenti nel database.

```
public Cursor getAllRecords(){
    SQLiteDatabase db=dbhelper.getReadableDatabase();

    return db.rawQuery("SELECT * FROM "+ DatabaseStrings.TBL_NAME,null);
}
```

La classe DbManager ci fornisce quindi i metodi per inserire, eliminare o selezionare promemoria senza esporre la struttura e le operazioni proprie del database.

ESEMPI

Promemoria • Ora possiamo utilizzare la classe appena creata per implementare l'interfaccia grafica che consenta all'utente di visualizzare tutti i promemoria, eliminarne uno e aggiungerne di nuovi. Il nostro layout è composto da:
- ListView, per visualizzare i titoli dei promemoria;
- EditText per permettere all'utente di inserire titolo e descrizione del nuovo promemoria;

- Button per inserire il nuovo promemoria all'interno del database.

Vediamo la parte di codice che ci permette di recuperare informazioni dal database e inserirle nella ListView.
Per permettere la comunicazione col database utilizziamo una variabile globale di tipo DbManager che inizializziamo nell'onCreate. D'ora in poi utilizziamo questo oggetto per ogni comunicazione con il database.
Per fare in modo che, all'apertura dell'applicazione, la ListView mostri tutti i titoli dei promemoria memorizzati nel database, dobbiamo far caricare i valori nella ListView all'interno del metodo onCreate della activity. Per riempire la ListView utilizzeremo un ArrayAdapter, in questo caso inizializzato con i titoli dei vari promemoria, per poi collegarlo alla ListView stessa.

MainActivity.java ×

```
package com.example.emanuele.promemoria;

import ...

public class MainActivity extends AppCompatActivity {

    ListView list = null;
    EditText titolo=null,descrizione=null;
    DbManager db =null;

    ArrayAdapter<String> adapter=null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        list = (ListView)findViewById(R.id.listView);
        titolo= (EditText)findViewById(R.id.editTextTitolo);
        descrizione=(EditText)findViewById(R.id.editTextDescrizione);

        db=new DbManager(this);

        Cursor c = db.getAllRecords();
```

```
        adapter = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1);
        while (c.moveToNext()){
            adapter.add(c.getString(c.getColumnIndex(DatabaseStrings.FIELD_TITOLO)));
        }
        list.setAdapter(adapter);

        ...

    }
```

Sempre all'interno del metodo onCreate andiamo a istanziare listener necessari per la gestione degli eventi onItemClick e onItemLongClick, associati agli elementi della ListView.

DEFINIZIONE

Il primo evento rappresenta il tap, cioè il tocco sullo schermo touch dell'elemento della ListView, mentre il secondo rappresenta il tocco prolungato (long press) sullo schermo touch dell'elemento della ListView.

Vediamo i due casi nello specifico:

- onItemClick: utilizziamo la posizione dell'elemento all'interno della ListView e il Cursor ottenuto dal metodo getAllRecords della classe DbManager per recuperare ulteriori informazioni sul promemoria.
  Una volta recuperata la descrizione del promemoria la visualizziamo tramite un Toast.

```
list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Cursor c = db.getAllRecords();
        c.moveToPosition(position);

        Toast.makeText(MainActivity.this,
                c.getString(c.getColumnIndex(DatabaseStrings.FIELD_DESCRIZIONE)),
                Toast.LENGTH_LONG).show();
    }
});
```

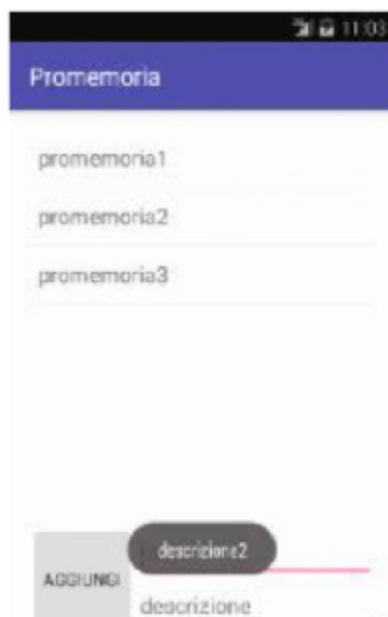
- onItemLongClick: recuperiamo l'id del promemoria in questione, utilizziamo il metodo delete della classe DbManager per rimuovere il promemoria dal database. Rimuoviamo il promemoria anche dall'ArrayAdapter della ListView e invochiamo il metodo notifyDataSetChanged affinché il contenuto della ListView si aggiorni.

```
list.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id){
        Cursor c = db.getAllRecords();
        c.moveToPosition(position);
```

```
            int idR = c.getInt(c.getColumnIndex(DatabaseStrings.FIELD_ID));
            adapter.remove(c.getString(c.getColumnIndex(DatabaseStrings.FIELD_TITOLO)));
            adapter.notifyDataSetChanged();
            db.delete(idR);

            return true;
        }
    });
```

Definiamo infine la funzione associata all'evento onClick del Button. La funzione addNew legge i valori del titolo e della descrizione dalle EditText e inserisce una nuova tupla all'interno della tabella promemoria. Infine aggiunge un nuovo elemento all'ArrayAdapter e richiama il metodo notifyDataSetChanged per aggiornare la ListView.

```
    public void addNew(View v){
        db.save(titolo.getText().toString(),descrizione.getText().toString());
        adapter.add(titolo.getText().toString());
        adapter.notifyDataSetChanged();
    }
```

Vediamone il funzionamento in esecuzione: nel database vi sono tre promemoria, l'utente ha appena fatto clic sulla seconda voce del promemoria per vederne la descrizione: ▶

ESEMPI

Rubrica Telefonica • Utilizziamo quanto appreso finora per realizzare un applicativo che consenta di memorizzare le informazioni dei contatti telefonici. I campi della tabella principale del database sono: nome, cognome e numero di telefono. La parte principale dell'applicazione riguarda la gestione delle informazioni all'interno del database.

Prima di tutto vediamo la struttura della tabella, composta in questo caso da quattro campi:

- ID: di tipo int identifica univocamente il contatto all'interno della tabella. Questa scelta è obbligata in quando potremmo avere tra i contatti più persone con lo stesso nome e cognome;
- Nome e Cognome: entrambi di tipo varchar per memorizzare le generalità del contatto;
- Numero: di tipo varchar rappresenta il numero telefonico del contatto.

In questo esempio la classe Person contiene i nomi dei campi.

Supponendo di disporre di un database già popolato vediamo come recuperare l'ID di un contatto a partire da cognome e nome. Utilizzando il metodo query() della classe SQLiteDatabase possiamo recuperare i soli record che soddisfano una determinata condizione. Il codice che segue mostra la funzione getID() che riceve il nome e cognome da cercare nella tabella, quindi dopo aver dichiarato la stringa selection che contiene la query SQL, esegue il metodo query che effettua la ricerca del nome nella tabella stessa e restituisce i risultati in un oggetto di classe cursor chiamato results:

```
public int getID(String nome, String cognome) {
    //recupero il riferimento al database in LETTURA
    SQLiteDatabase db = dbhelper.getReadableDatabase();

    //creo la stringa di selezione
    String selection = Person.FIELD_NOME + "=? AND " + Person.FIELD_COGNOME + "=?";

    //creo il vettore di string necessario per passare i parametri come argomenti della query
    String[] selectionArgs = {nome, cognome};

    //creo il vettore di string necessario per la selezione delle colonne
    String[] columns = {Person.FIELD_ID};

    //utilizzo il metodo insert della classe SQLiteDatabase per inserire il nuovo contatto
    Cursor results = db.query(Person.TBL_NAME, columns, selection, selectionArgs,
            null, null, null, null);

    //verifico se l'aggiornamento è avvenuto con successo
    if (results.moveToNext()) {
        return results.getInt(0);
    } else {
        return -1;
    }
}
```

Il metodo query effettua una query SQL utilizzando sostanzialmente quattro parametri:

- tabella su cui operare (Person.TBL_NAME);
- colonne sulle quali effettuare la ricerca (columns);
- colonne da ottenere come risultato (selectionArgs);
- stringa di selezione da includere nelle where clause (selection).

In pratica la query descritta equivale a:

```
SELECT nome, cognome
FROM Person.TBL_NAME
WHERE FIELD_NOME=nome AND FIELD_COGNOME=cognome
```

Dove nome e cognome della where clause sono le variabili ricevute come parametro dalla funzione, sostituite nella stringa di selezione dal carattere punto di do-

La chiave primaria della tabella è rappresentata dal campo ID, pertanto se facciamo una selezione per nome e cognome potremmo ottenere come risultato più di un ID, poiché non è garantita l'univocità della coppia nome, cognome.

manda (?), mentre invece nome e cognome poste accanto alla SELECT sono quelle che verranno collocate nell'oggetto results di classe Cursor. Possiamo utilizzare l'ID ottenuto anche per eliminare il record relativo al contatto selezionato. Per fare ciò dobbiamo utilizzare il metodo delete della classe SQLiteDatabase e specificare, utilizzando la where clause, che vogliamo eliminare solo il record avente l'ID specificato.

```
public boolean delete(int id) {
    //recupero il riferimento al database in SCRITTURA
    SQLiteDatabase db = dbhelper.getWritableDatabase();

    //creo il vettore di string necessario per passare i parametri come argomenti della query
    String[] whereArgs = {String.valueOf(id)};

    //utilizzo il metodo delete della classe SQLiteDatabase per eliminare il contatto
    long numRowsDeleted = db.delete(Person.TBL_NAME, Person.FIELD_ID + "=?", whereArgs);

    //verifico se l'eliminazione è avvenuta con successo
    if (numRowsDeleted > 0) {
        return true;
    } else {
        return false;
    }
}
```

Per inserire un nuovo contatto all'interno del database dobbiamo utilizzare il metodo insert, passando come parametro un oggetto ContentValue creato utilizzando le informazioni del nuovo contatto.

```
public boolean insert(String nome, String cognome, String numero) {
    //recupero il riferimento al database in SCRITTURA
    SQLiteDatabase db = dbhelper.getWritableDatabase();

    //creo un nuovo oggetto ContentValue
    ContentValues cv = new ContentValues();

    //riempio il content value con i valori relativi al contatto che voglio inserire
    cv.put(Person.FIELD_NOME, nome);
    cv.put(Person.FIELD_COGNOME, cognome);
    cv.put(Person.FIELD_NUMERO, numero);

    //utilizzo il metodo insert della classe SQLiteDatabase per inserire il nuovo contatto
    long id = db.insert(Person.TBL_NAME, null, cv);

    //verifico se l'inserimento è avvenuto con successo
    if (id == -1) {
        return false;
    } else {
        return true;
    }

}
```

Nel caso in cui un nostro contatto cambiasse numero di telefono sarebbe molto utile per noi disporre di una funzione di aggiornamento, che permette di modificare il numero di telefono di un determinato contatto.

```
public boolean update(int id, String numero) {
    //recupero il riferimento al database in SCRITTURA
    SQLiteDatabase db = dbhelper.getWritableDatabase();

    //creo il vettore di string contenete i parametri della whereClause
    String[] whereArgs = {String.valueOf(id)};

    //creo un nuovo oggetto ContentValue
    ContentValues cv = new ContentValues();

    //riempio il content value con i valori relativi al contatto che voglio modificare
    cv.put(Person.FIELD_NUMERO, numero);

    //utilizzo il metodo insert della classe SQLiteDatabase per inserire il nuovo contatto
    long numRowsUpdated = db.update(Person.TBL_NAME, cv, Person.FIELD_ID + "=?", whereArgs);

    //verifico se l'aggiornamento è avvenuto con successo
    if (numRowsUpdated > 0) {
        return true;
    } else {
        return false;
    }
}
```

Quando utilizziamo il metodo update l'oggetto ContentValues deve contenere solo le coppie attributo valore da modificare. Nel caso appena citato non è nostro interesse modificare nome e cognome del contatto, quindi includiamo nel ContentValues solo il numero di telefono.

Prova adesso!

- Utilizzare il metodo query
- Applicare i metodi: insert, delete, update

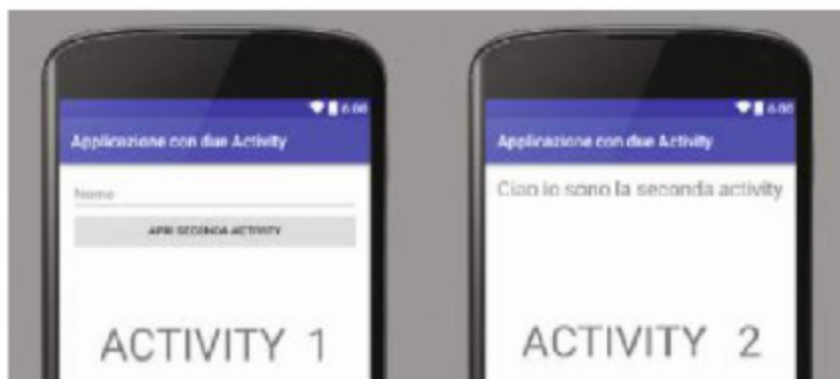
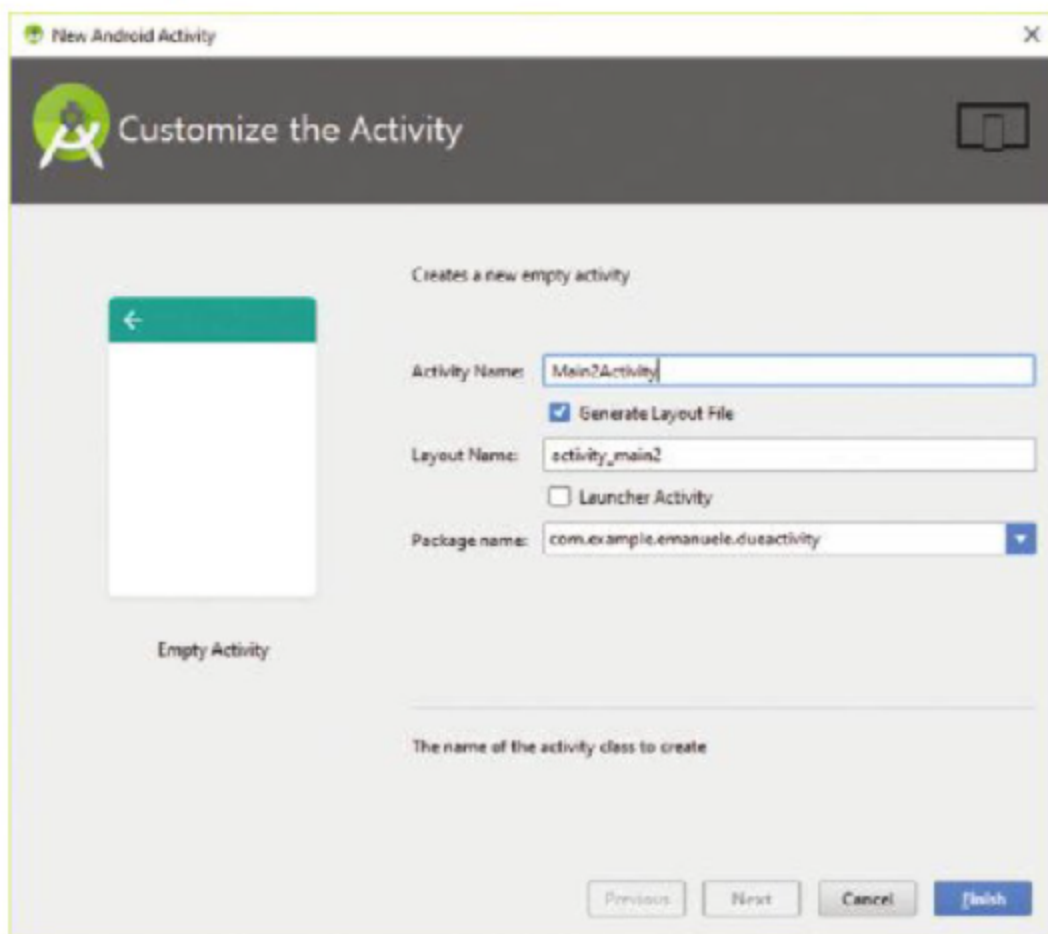
Apri l'esempio Rubrica telefonica

1. Modifica il codice in modo che per ogni contatto si possa memorizzare anche email e un secondo numero di telefono.
2. Modifica il codice aggiungendo una funzione che verifica se il contatto esiste, e nel caso esista, restituisca il numero di telefono a esso associato.
3. Modifica il codice in modo che l'utente possa visualizzare la lista dei contatti il cui nome è Marco.

Intent

Quando la complessità dell'applicazione aumenta può essere necessario organizzare la stessa dividendone gli aspetti funzionali in diverse activity. Vediamo come realizzare un'applicazione composta da due activity. Per prima cosa creiamo un nuovo progetto

con una attività vuota (empty activity), quindi aggiungiamo una seconda activity alla nostra applicazione selezionando da menu File → New → Activity → Empty Activity. Scegliamo il nome della activity, mettiamo la spunta su Generate Layout File e facciamo clic su Finish.

Spuntando Launcher Activity l'activity appena creata diventa l'activity di avvio dell'applicazione.

Android Studio aggiungerà due nuovi file al progetto:
- Main2Activity.java che contiene il codice della seconda activity;
- activity_main2.xml che contiene il layout della seconda activity.

Modifichiamo i layout delle activity come mostrato dalla videata riportata in fondo alla pagina precedente. Utilizziamo l'evento onClick sul pulsante APRI SECONDA ACTIVITY per mandare in esecuzione la seconda activity. Per far ciò creiamo un nuovo oggetto Intent, specificando la nuova activity da avviare, chiamando il metodo startActivity e passando l'Intent appena creato come parametro, come indicato dal codice seguente:

```
package com.example.emanuele.dueactivity;
import ...
public class MainActivity extends AppCompatActivity {
    EditText nome = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        nome=(EditText)findViewById(R.id.editText);
    }
    public void buttonClick(View v){
        Intent i=new Intent(MainActivity.this,Main2Activity.class);
        startActivity(i);
    }
}
```

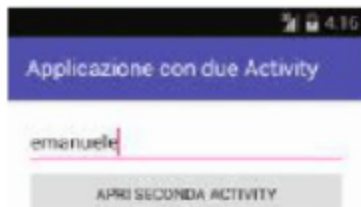
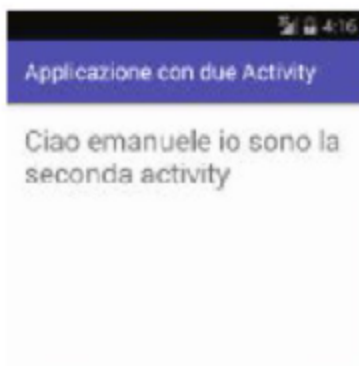
L'Intent non serve solo per avviare una nuova activity, ma è un oggetto vero e proprio con lo scopo di fare da interfaccia tra le due activity, possiamo usare questo oggetto anche per passare alcuni parametri tra le activity.
Per inviare alla seconda activity il testo inserito dall'utente nella EditText, prima di chiamare il metodo startActivity, aggiungiamo una nuova coppia chiave valore all'Intent, utilizzando il metodo putExtra.

```
public void buttonClick(View v){
    Intent i=new Intent(MainActivity.this,Main2Activity.class);
    i.putExtra("nome",nome.getText().toString());
    startActivity(i);
}
```

All'interno dell'onCreate della seconda activity recuperiamo l'oggetto Intent tramite il metodo getIntent e leggiamo il valore associato alla chiave specificata grazie al metodo getStringExtra. Modifichiamo infine il testo della TextView collocandovi il testo appena letto.

Il metodo utilizzato per la lettura del valore varia a seconda del tipo di dato che deve essere ricevuto.

```
package com.example.emanuele.dueactivity;
import ...
public class Main2Activity extends AppCompatActivity {
    TextView text = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        text=(TextView)findViewById(R.id.textView);
        Intent i = getIntent();
        String nome = i.getStringExtra("nome");
        text.setText("Ciao " + nome + " io sono la seconda activity");
    }
}
```

Verifichiamo il funzionamento dell'applicazione. Inseriamo il nostro nome nella EditText e facciamo clic sul pulsante APRI SECONDA ACTIVITY: ▶

A questo punto viene eseguita la seconda activity, che riceve il messaggio dalla EditText della prima activity e lo mostra nella TextView dopo averlo leggermente rielaborato: ▶

Prova adesso!

Apri l'esempio Promemoria

▸ Utilizzare la classe Intent

1. Modifica l'esempio promemoria utilizzando una seconda activity per l'inserimento di nuovi promemoria.
2. Modifica l'esempio aggiungendo una nuova activity per visualizzare i dettagli del promemoria selezionato.

Esercitiamoci

Test Vero/Falso

Indica, barrando la relativa casella, se le seguenti affermazioni sono vere o false.

1 I metodi per creare e aggiornare il database sono implementati nella classe SQLiteOpenHelper. V F
2 È possibile creare solo un database. V F
3 Il metodo onCreate viene eseguito ogni volta che il database viene aperto. V F
4 È buona norma salvare i nomi delle tabelle e degli attributi in un'altra classe. V F
5 È possibile ottenere un riferimento al database in sola lettura. V F
6 Con il metodo getWritableDatabase non si possono effettuare operazioni di lettura. V F
7 La classe SQLiteDatabase permette di inserire nuovi record nel database. V F
8 Ogni metodo della Classe SQLiteDatabase ritorna un oggetto di tipo Cursor. V F
9 È possibile utilizzare un Cursor per modificare dati all'interno del database. V F
10 Cursor è la classe che fornisce accesso al result set di una query. V F
11 Per inserire nuovi dati nel database si deve utilizzare un oggetto Cursor. V F
12 La classe ContentValues rappresenta una sequenza di coppie chave/valore. V F
13 Ogni oggetto di classe ContentValues è in formato read only. V F
14 La classe Intent permette di passare parametri all'activity destinataria V F

Domande a risposta multipla

Indica la risposta corretta barrando la casella relativa.

1 Quale metodo permette alla ListView di aggiornare il proprio contenuto in seguito a una modifica dell'ArrayAdapter?

a. add
b. remove
c. notifyDataSetChanged
d. nessuno dei precedenti

2 Il metodo query della classe SQLiteDatabase restituisce un oggetto di tipo:

a. non restituisce alcun valore
b. Cursor
c. int
d. ContentValues

3 All'interno di quale metodo è opportuno effettuare un controllo sulla versione del database?

a. onCreate
b. onOpen
c. onUpdate
d. onConfigure

4 Quale dei seguenti metodi della classe ContentValues permette di inserire una nuova coppia chiave/valore?

a. getAsString
b. getAsInt
c. valueSet
d. put

5 Il metodo execSQL della classe SQLiteDatabase restituisce un oggetto di tipo:

a. non restituisce alcun valore
b. Cursor
c. int
d. ContentValues

Esercitiamoci

Problemi

Progetta e realizza completamente il codice in Java per Android che risolva il problema proposto.

1 Crea un progetto per la gestione dei contatti email dell'utente, registrando la data di inserimento, l'email stessa e il nome del contatto o dell'azienda a essa associata.

2 Crea un progetto che implementi una semplice galleria fotografica. L'utente può assegnare a ogni immagine un titolo e una descrizione. Deve inoltre essere possibile eliminare o aggiungere nuove immagini alla galleria.

3 Crea un progetto per la gestione della lista della spesa. L'utente deve poter eliminare e aggiungere elementi alla lista. L'applicazione deve consentire all'utente la gestione di più liste.

4 Crea un progetto per la gestione di una biblioteca. L'utente deve poter inserire o modificare le schede relative ai libri. Deve inoltre essere presente una funzione di ricerca di un libro basata su titolo e autore.

5 Crea un progetto per la gestione degli scontrini. L'applicazione deve memorizzare per ogni scontrino la data, il totale dell'importo e l'eventuale lista di prodotti o servizi acquistati. L'utente deve poter visualizzare la cronologia degli acquisti e il totale delle spese giornaliere, mensili e annuali.

Scheda di autovalutazione

Conoscenze	Scarso	Medio	Ottimo
Riconoscere gli elementi di un database Android SQLite	○	○	○
Individuare le differenze tra dati privati e pubblici	○	○	○
Conoscere i metodi della classe SQLiteOpenHelper	○	○	○
Conoscere i metodi della classe SQLiteDatabase	○	○	○
Comprendere il significato di Preferences	○	○	○
Riconoscere il ruolo dell'annotazione @Override	○	○	○
Conoscere il ruolo della classe Cursor	○	○	○
Conoscere il ruolo della classe Intent	○	○	○
Conoscere il ruolo della classe ContentValues	○	○	○

Competenze	Scarso	Medio	Ottimo
Creare applicazioni che utilizzino la classe SQLiteOpenHelper	○	○	○
Creare un database SQLite locale	○	○	○
Applicare la classe SQLiteDatabase	○	○	○
Utilizzare il metodo onCreate in override	○	○	○
Utilizzare il metodo onUpgrade in override	○	○	○
Applicare il metodo getReadableDatabase	○	○	○
Applicare il metodo getWritebleDatabase	○	○	○
Applicare la classe Cursor	○	○	○
Applicare la classe Intent	○	○	○
Applicare la classe ContentValues	○	○	○